Fall 1980

A Cross Assembler for the M6800 Microprocessor

Donald J. Walsh
University of Central Florida

A CROSS ASSEMBLER FOR THE M6800 MICROPROCESSOR

BY

DONALD J. WALSH, JR.
B.A., TRENTON STATE COLLEGE, 1975

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in the
Graduate Studies Program of the
College of Engineering at the
University of Central Florida;
Orlando, Florida

Fall Quarter
1980

ABSTRACT

This research paper describes a cross assembler for the Motorola 6800 assembly language which was written in Fortran to run on the Digital Equipment Corporation PDP 11/34.

The assembler provides an assembly listing on the CRT or Printer and an Object File. In addition, the assembler will check for errors during PASS 1 and also during PASS 2.

TABLE OF CONTENTS

ILLUSTRATIONS

Figure

ACKNOWLEDGEMENT

First, I would like to thank my graduate committee; and, in particular, my advisor, Dr. Bauer, for his guidance and encouragement during my course of study at UCF and also in the preparation of this research report.

I would also like to thank the management at NTEC for the encouragement and tuition aide program, without which I would not have been able to complete my course work.

In addition I would like to thank my wife and typist Trudi whose patience and encouragement proved to be an invaluable asset.

Finally, I would like to thank my children Patrick and Tara who will have a father once again.

INTRODUCTION

There are two ways to convert source code (Motorola 6800 assembly mnemonic code) to machine language. One way is to hand assemble it using a manufacturer's programming manual. The second method is to use an assembler.[1] Hand assembly is a time-consuming and tedious task. In addition, hand assembly provides no error checks. An assembler will provide object code quickly in addition to checking for errors.

The assembler described in this report translates source assembly language mnemonic statements and symbolic statements into machine language instruction for the M6800 assembly language.[2]

Figure 1 shows a block diagram of the assembler. The source file is defined by the user and is input to the assembler which outputs an object file and an assembly listing.

The assembler is a two pass assembler. The first pass creates a symbol table by reading each line of code and determining if the line has a label and then determining the value for the label. In addition, the first pass must determine whether the instruction is a 1, 2, or 3 byte instruction in order to increment the location counter appropriately.

PASS 1 also provides an error check for the existence of the same label in symbol table for a previous instruction. In

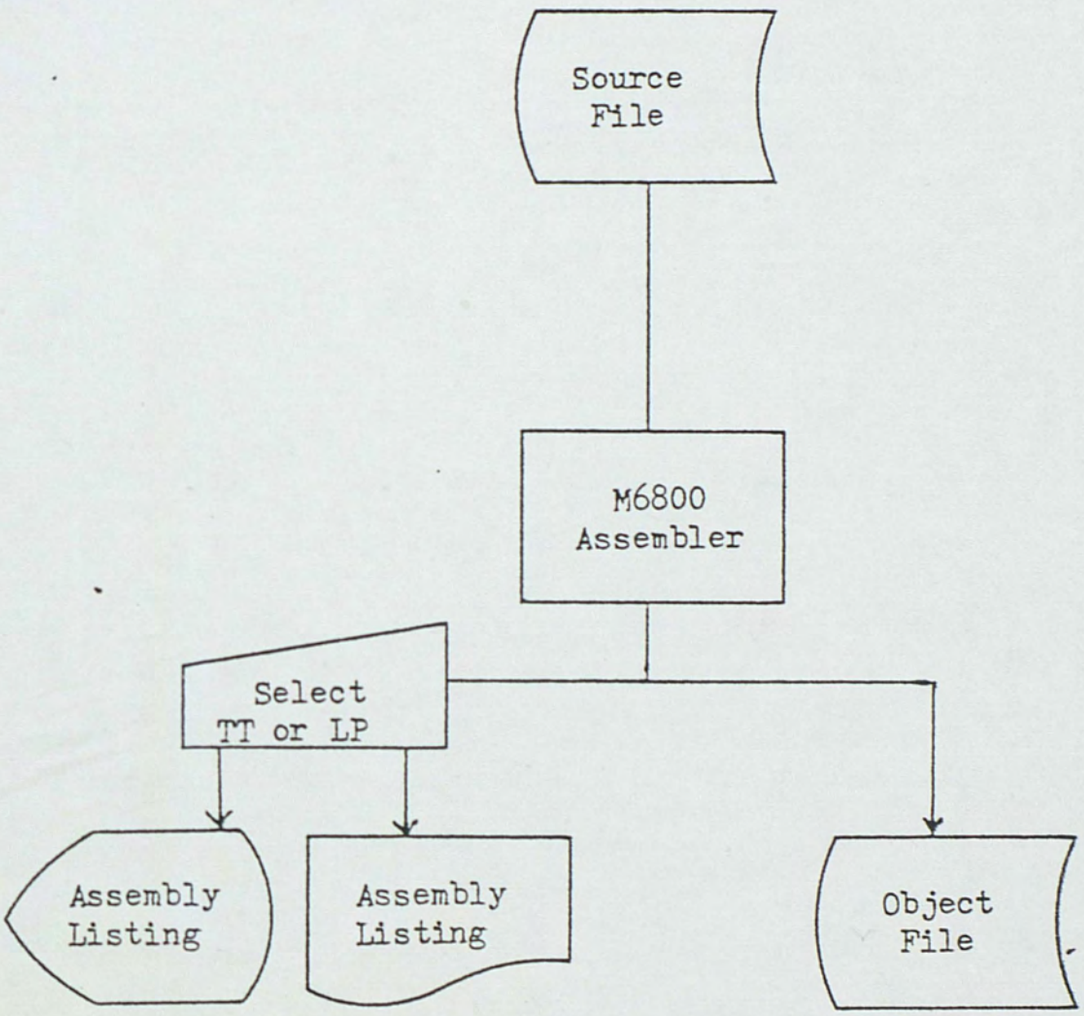

Fig. 1. M6800 Assembler block diagram

addition, it checks to see if the label meets the criteria for a label (i.e., first character is letter, only six alphanumeric characters long, and is not the single letter X, A, or B). PASS 1 is terminated when an END order is encountered.

The purpose of PASS 2 of the assembler is to convert the source language into HEX code. It does this by using the symbol table constructed in PASS 1 to convert the symbols to their HEX value and the mnemonic table to convert the instructions. PASS 2 then evaluates the operand to determine its HEX value. To do this, the source program is read a line at a time and many steps of PASS 1 repeated. PASS 2 creates two outputs an assembly listing and an object listing. PASS 2 performs error checking and errors are printed with the assembly listing. PASS 2 is also terminated when the END pseudo order is read.

The purpose of this paper is to describe the organization of an assembler for the Motorola M6800 assembly language. The assembler is written in Fortran to run on the PDP 11/34. A user's guide is provided in Appendix A.

CHAPTER I

M6800 ASSEMBLER STRUCTURE

Before the M6800 Assembler can be run, a source file must be created using the RT11 Editor.

A sample file DK1:M683.ASS is provided in Appendix A. Once the M6800 Assembler is run it provides an assembly listing on the CRT or printer, whichever the user selects. It also creates an object file and places this file on the same disk system as the source file. The object file for DK1:M683.ASS would be DK1:M683.OBJ. A sample object file is provided in Appendix A.

The M6800 Assembler was written in Fortran to run on the PDP11/34. The program consists of a main program with 28 subroutines. The main program and 28 subroutines were compiled separately using the NOVECTORS, NOLINENUMBERS, NOOPTIMIZE:SPD options of the RT11 Fortran compiler. These options decrease the run time speed of the program but allow the program to run using less memory. The object files were then linked together to form a save file called M6800.SAV.

The assembler is a two pass assembler. The first pass analuzes each line of source code for a label. If it finds a valid label that is not already defined, it assigns it a value and stores it in the symbol table. PASS 1 must also keep track

of the location counter; therefore, it has to cetermine if each instruction is a 1, 2, or 3 byte instruction and increment the location counter appropriately.

PASS 2, in addition to keeping track of the location counter, also finds the HEX value for the operator field and the operand field. It evaluates each source line and outputs the equivalent assembly listing code and also supplies the object subroutine with inputs for the object file.

To run the M6800 Assembler, simply type RUN M6800 after the RT11 period prompt. The program will be loaded into memory and will prompt the user to enter the logical unit number for the line printer or the CRT. This allows for the option of having the assembly listing displayed on the CRT or printed on the line printer. The program then asks for the name of the source file. This is all done in the main program.

The operator field can contain one of the eight assembler directives, or the operator field can have one of 197 possible HEX values. The operand is evaluated by searching the opcode table (table of mnemonics) for the appropriate HEX code. This is done by supplying a subroutine with a 5 byte variable called OPS. The first 3 bytes of OPS are the first three letters of the operand. The fourth byte is the "A" for accumulator A or "B" for accumulator B or "SP" for no accumulator. The last byte contains a code for the mode of addressing. There are six modes of addressing. These are listed below with the value input into the last byte of OPS.

"I" = immediate "D" = direct "N" = index

"E" = extended "H" = inherent "R" = relative

An opcode table is provided in Appendix B.

The operand field can contain the following:

1. Hexadecimal number
2. Octal number
3. Binary number
4. Decimal number
5. ASCII literal character
6. * indicating location counter

In addition, the M6800 Assembler provides the following error checks:

1. Invalid opcode
2. Symbol not found in symbol table
3. Not valid symbol
4. Invalid opcode, use of accumulator, or addressing mode
5. Only character of ASCII character set allowed
6. Branch out of bounds
7. Operand field greater than 255
8. Operand field greater than 65535
9. Operand not decimal number
10. Operand not octal number
11. Operand not binary number
12. Operand not HEX number
13. No operand

14. Count greater than 255

15. Address greater than FFFF

16. Symbol already assigned value

CHAPTER II

M6800 ASSEMBLER CODING

The M6800 Assembler consists of a main program and 28 subroutines. The subroutines allow for a modular approach to development and ease in debugging.

Labeled common was used, and a list of the variables and their definitions are provided below:

HEXB: a 15 by 4 integer array used for HEX to binary conversion.

BUFFER: an 80 byte array used to store a line of source code.

SYMA: a 100 by 4 byte array to store addresses for symbol table.

SYMT: a 100 by 6 byte array to store names of symbols for symbol table.

LOCC: a 4 byte array to store HEX value to location counter.

LOCD: real number used to store decimal value of location counter.

IFLAG: used as flag in search subroutine to determine if symbol table needs to be alphabetized before a search.

OPTAB: a 197 by 5 byte array contains opcode table shown in Appendix B.

OPHEX: a 197 by 2 byte array contains the HEX value for the corresponding opcode as depicted in Appendix B.

OBJF: a 42 byte array stores one line of object code.

NEND: Flag for end of source listing.

NSTART: Flag for start of source listing.

NORG: Flag for call from origin subroutine.

NUMBER: a real number used to sum object HEX values for error check.

NSTOBJ: keeps track of length of line of object file.

ERROR: array contains list of errors.

NERR: contains number of errors found.

KWRI: logical unit number for writing assembly listing, either to CRT or printer.

A description of the main program and the 28 subroutines is provided below.

M6800 Assembler Main Program

Figure 2 provides a block diagram for the main program. The main program first prompts for the logical unit number for the line printer or CRT. It then prompts for the file name for the source file, and then it creates the name for the object file. After opening the source file, it reads the first record and calls subroutine PASS1. After the complete source file has been read and PASS1 has created a symbol table, the source file is closed. If there were no errors in PASS1, the source file is open again and PASS2 subroutine is called. Opening and closing the source file performs the same function as rewinding the file. The PASS2 subroutine provides an assembly listing of the source file on either the line printer or CRT and provides an object file on the same disk system that the source file is on. PASS1 passes the variable M to the main program, which is then passed to PASS2 from the main program. M represents the length of the

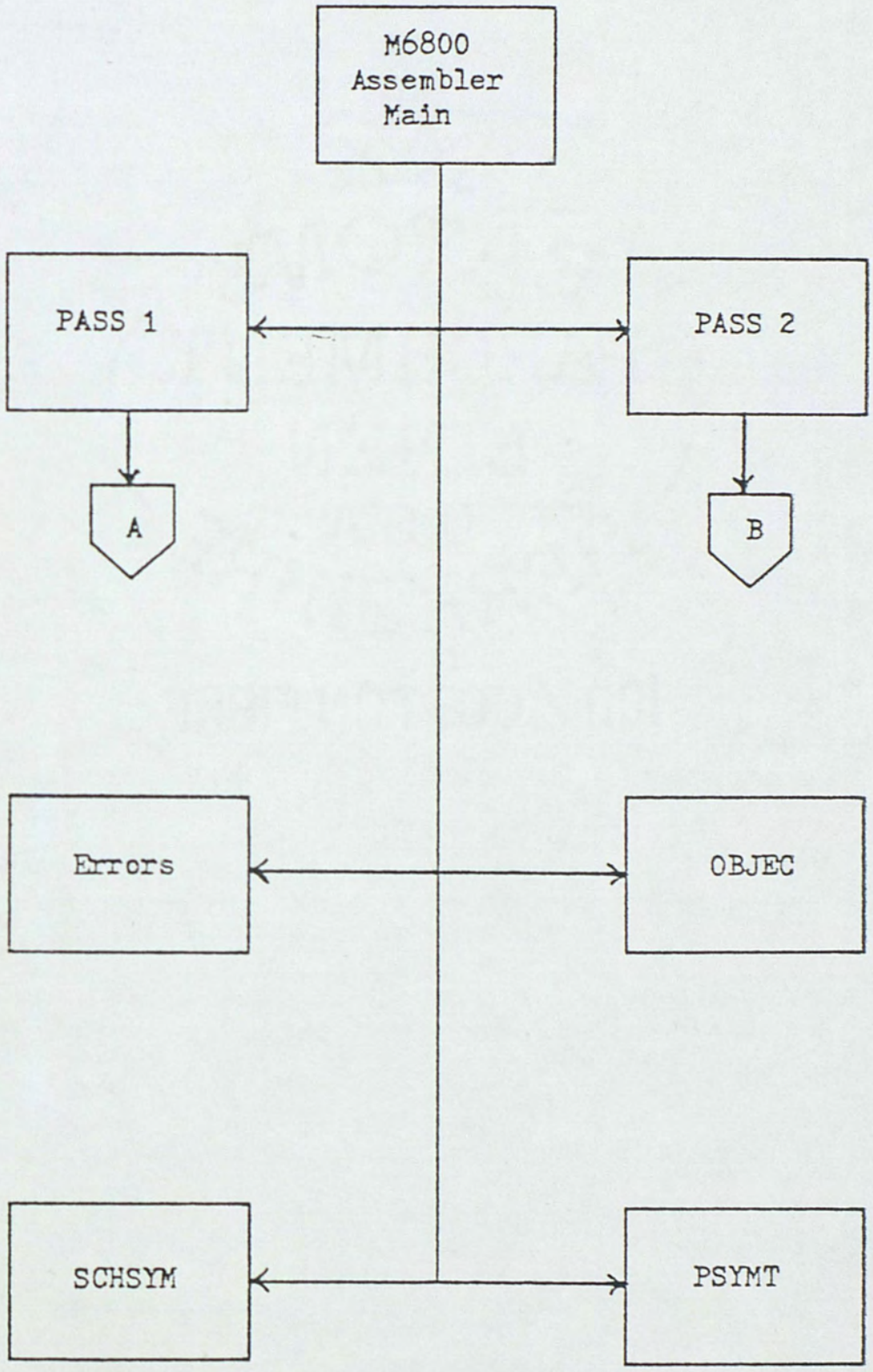

Fig. 2. M6800 Assembler main block diagram

symbol table plus one.

After the complete source file has gone through PASS2, and if there are no errors, OBJEC is called to write the last line of the object file to the disk. Then SCHSYM subroutine is called to alphabetize the symbol table, if this hasn't been done already. Next, the PSYMT subroutine is called to print the symbol table. The main program then informs the user where the object file can be found. If there are errors in either PASS1 or PASS2, the ERRORS subroutine is called and prints out a list of the errors for the line of the source code that created the error condition. If errors are found in the first PASS, the program will stop right after PASS1 has been completed for the entire source file. Before the program stops, the source file and object file are closed.

PASS1

PASS1 first checks for no label. This is done by checking for two spaces or a TAB in the first two positions of the BUFFER array. If a label is found, it is stored in the SYMT array. The label is then examined to see if it is a valid label by calling the CKLABL subroutine, as depicted in Figure 3. Then SCHSYM is called to see if the label already exists. PASS1 then evaluates the operator to see if it is the assembler directive EQU. If it is, the EQU subroutine is called to evaluate the operand and stores it in the SYMA array. If the operator is not EQU, SYMA is loaded with the location counter LOCC.

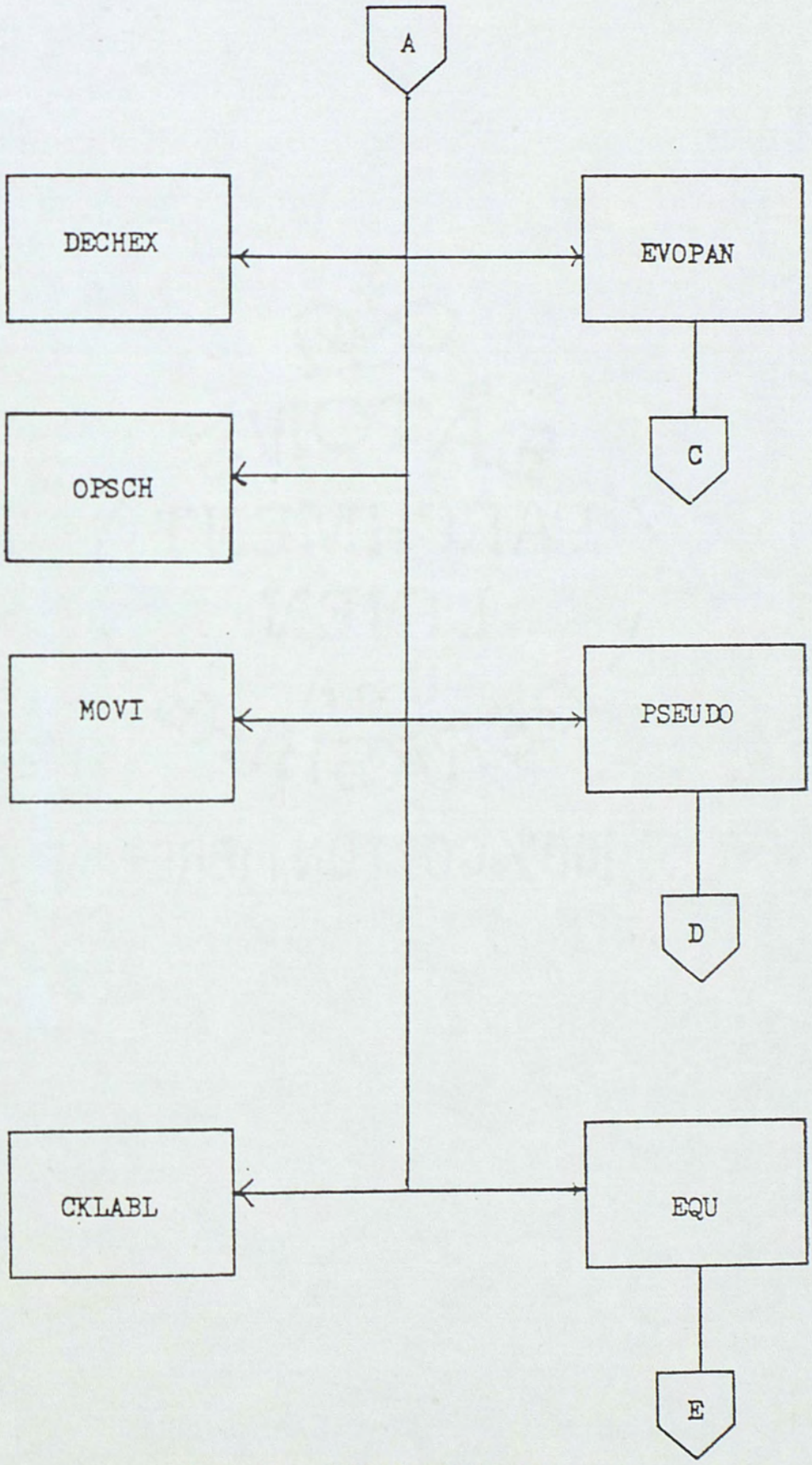

Fig. 3. PASS1 block diagram

If the assembler directive EQU was not found, the operator is checked for END. If END is found, the NEND flag is set and the subroutine returns control to the main program.

If an END was not encountered, PASS1 then evaluates the operator by calling the subroutine PSEUDO. If PSEUDO finds an assembler directive, it sets the IPSEU flag and returns control to the main program through PASS1.

If a pseudo op was not encountered, PASS1 then evaluates the operator and operand to determine if the instruction is a 1, 2, or 3 byte instruction. PASS1 does this by checking the first letter of the operator for a B and the second to see if it is not an I. This indicates that the instruction is a Branch instruction; and, therefore, the location counter is incremented by two. Control is then passed to the main program.

If PASS1 finds that the operator is not a Branch instruction, it loads the first three letters of the operator into the OPS byte array. It then determines if the operator uses accumulator A or B or neither. If it uses the A accumulator, an A is placed in OPS. Or if the B accumulator is used, a B is placed in OPS. If neither accumulator is used, a blank is placed in OPS. An H is then placed in the last position of the OPS variable, and OPSCH is called. The H represents inherent addressing, and OPSCH searches the OP CODE table for the variable in OPS. If a match is found, inherent addressing was used; and the location counter is incremented by one. If there was no

match, PASS1 then checks for immediate addressing by examining the first position of the operand for a # sign. If a # sign is found, the operator is examined to determine if it contains one of the following opcodes: CPX, LDS, or LDX. If one of these opcodes is found, the location counter is incremented by three; if not, it is incremented by two. If a # sign is not found, PASS1 then checks for index addressing. It does this by checking the operand for an X by itself or a ,X. If neither of these is found, the location counter is incremented by two. If an X or ,X is not found, the operand is evaluated by calling EVOPAN. If the decimal value of the operand is less than or equal to 255, the location counter is incremented by two; if not, the location counter in incremented by three. This represents direct and extended addressing, respectively. In each case above, incrementing the location counter involves incrementing LOCD and then calling DECHEX to insert the new HEX value of the location counter into LOCC. After the location counter has been incremented, control is returned to the main program.

PASS2

PASS2 first checks for no label. This is done by checking for two spaces or a TAB in the first two positions of the BUFFER array. If a label is found, PASS2 evaluates the operator to see if it contains the assembler directive EQU. If it does, the EQU subroutine is called, as depicted in Figure 4. Then on return from EQU, PASS2 returns control to the main program.

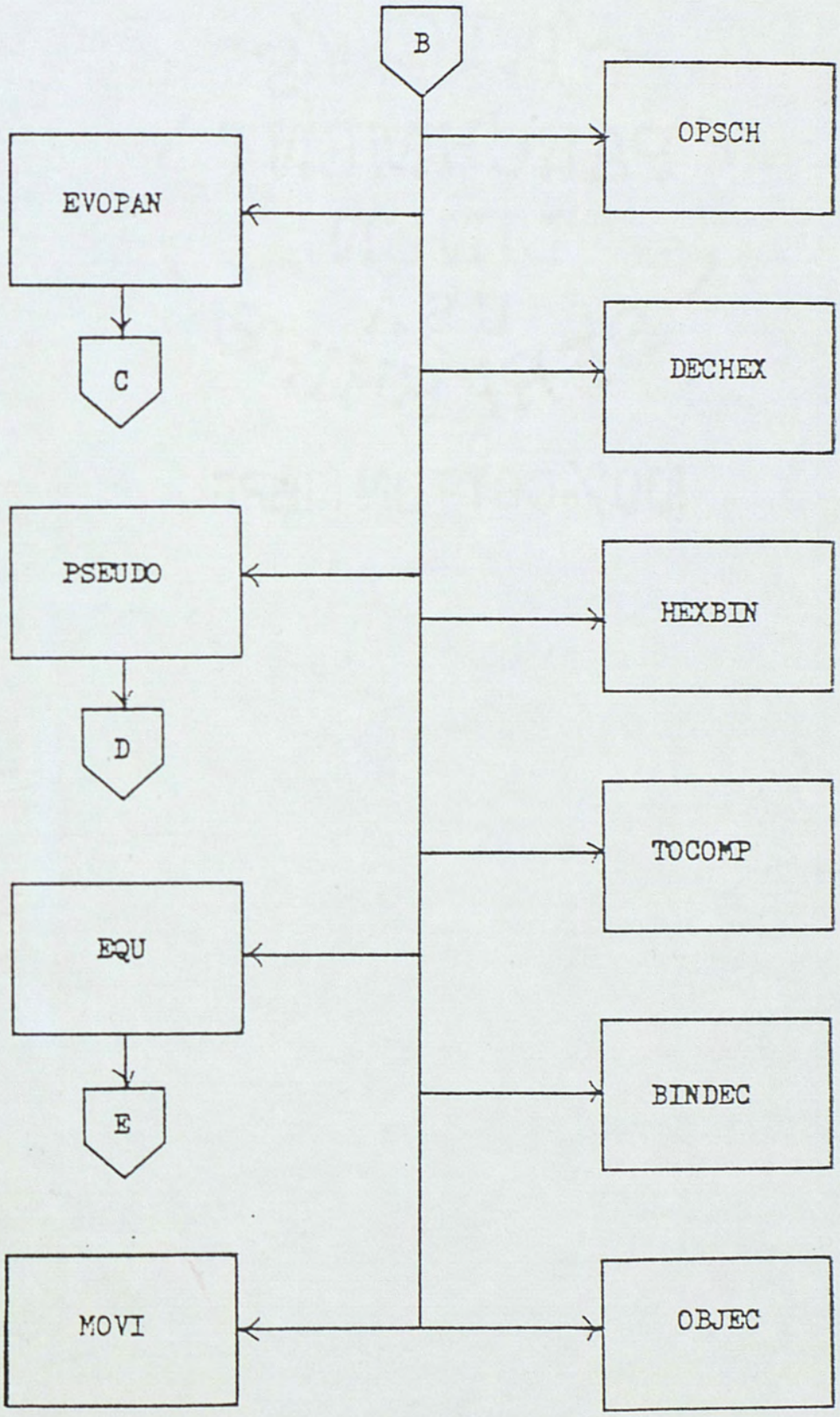

Fig. 4. PASS2 block diagram

If there is no label or the EQU assembler directive was not found, the operator is checked for the END statement. If END is found, the NEND flag is set; and the subroutine returns control to the main program.

If an END was not encountered, PASS2 then evaluates the operator by calling the subroutine PSEUDO. If PSEUDO finds an assembler directive, it sets the IPSEU flag, which indicates to PASS2 to return control to the main program.

If a pseudo op was not encountered, PASS2 examines the first letter of the operator for a B and the second to see if it is not an I. This indicates that the instruction is a Branch instruction. If this condition exists, the first three letters of the operator are loaded with a blank and the fifth with an "R". The "R" indicates relative addressing. OPSCH is called to determine the HEX code for this opcode. EVOPAN is called to evaluate the operand. The decimal value of the location counter plus two is subtracted from the decimal value of the operand. This value is then checked to see if it is less than -128 or greater than 127. If it is, an error message is selected. If the distance of the branch NDIS is in the valid range, it is then checked to see if the value is negative (backward branch). If the branch is a backward branch, NDIS is converted to binary. The twos complement is taken, and then this value is converted to HEX and represents the HEX code offset for this branch instruction. If NDIS is positive, it is converted to HEX and

represents the offset for the branch instruction. After the offset has been calculated, this line of the assembly listing is sent to the CRT or printer. Then the OBJT byte array is loaded with the appropriate HEX code and sent to subroutine OBJEC. On return from OBJEC, control is returned to the main program.

If the source line was not a branch instruction, the first three letters of the operator are placed into the OPS byte array. It then determines if the operator uses accumulator A or B or neither. If it uses the A accumulator, an A is placed in OPS. If the B accumulator is used, a B is placed in OPS. If neither is used, a blank is placed in OPS. An H is then placed in the last position of the OPS variable, and OPSCH is called. OPSCH searches the opcode table for the variable in OPS. If a match is found, inherent addressing was used; and the HEX code returned from OPSCH is used to write the assembly listing to the CRT or printer. OBJT is then loaded with this HEX code and OBJEC is called. After control returns to PASS2, it returns control to the main program. If inherent addressing was not used, PASS2 then checks for a # sign. If a # sign is found, the operator is examined to determine if it contains one of the following opcodes: CPX, LDS, LDX. If one of these opcodes is found, it indicates a 3 byte instruction using immediate addressing instead of a 2 byte instruction using immediate addressing. In either case, an I is placed in the fifth position of OPS, and OPSCH is called to determine the HEX code for the operator. The next

step is to call EVOPAN to evaluate the HEX code for the operand. Then the assembly listing is written to the CRT and printer. Next, OBJT is formed and OBJEC is called. Upon return from OBJEC, control is returned to the main program. The same procedure that was used in PASS1 is used in PASS2 to determine if the instruction used indexed, direct, or extended addressing. The same procedure as outlined above for immediate addressing is followed for the three modes of addressing mentioned above. If indexed addressing is used, the fifth position of OPS is loaded with N; for direct addressing, OPS is loaded with D; for extended addressing, OPS is loaded with E. Once the operand is evaluated, it is examined to see if it is greater than 255 or 65535, depending on the mode of addressing. An error code is generated, if the operand is greater than the appropriate value for the particular mode of addressing.

EVOPAN

The arguments for EVOPAN are: I, the position indicator for the BUFFER array; LENH, the length of the operand in HEX; NU, the decimal value of the operand; NSYS, the operand number system indicator; HEXV, contains the HEX value of the operand; M, contains the length of the symbol table plus one.

This subroutine first examines the operand to see if it contains a label. If it does, the LABVAL subroutine is called as depicted in Figure 5. LABVAL determines the value for the label in HEX, then the HEXDEC subroutine converts this to decimal.

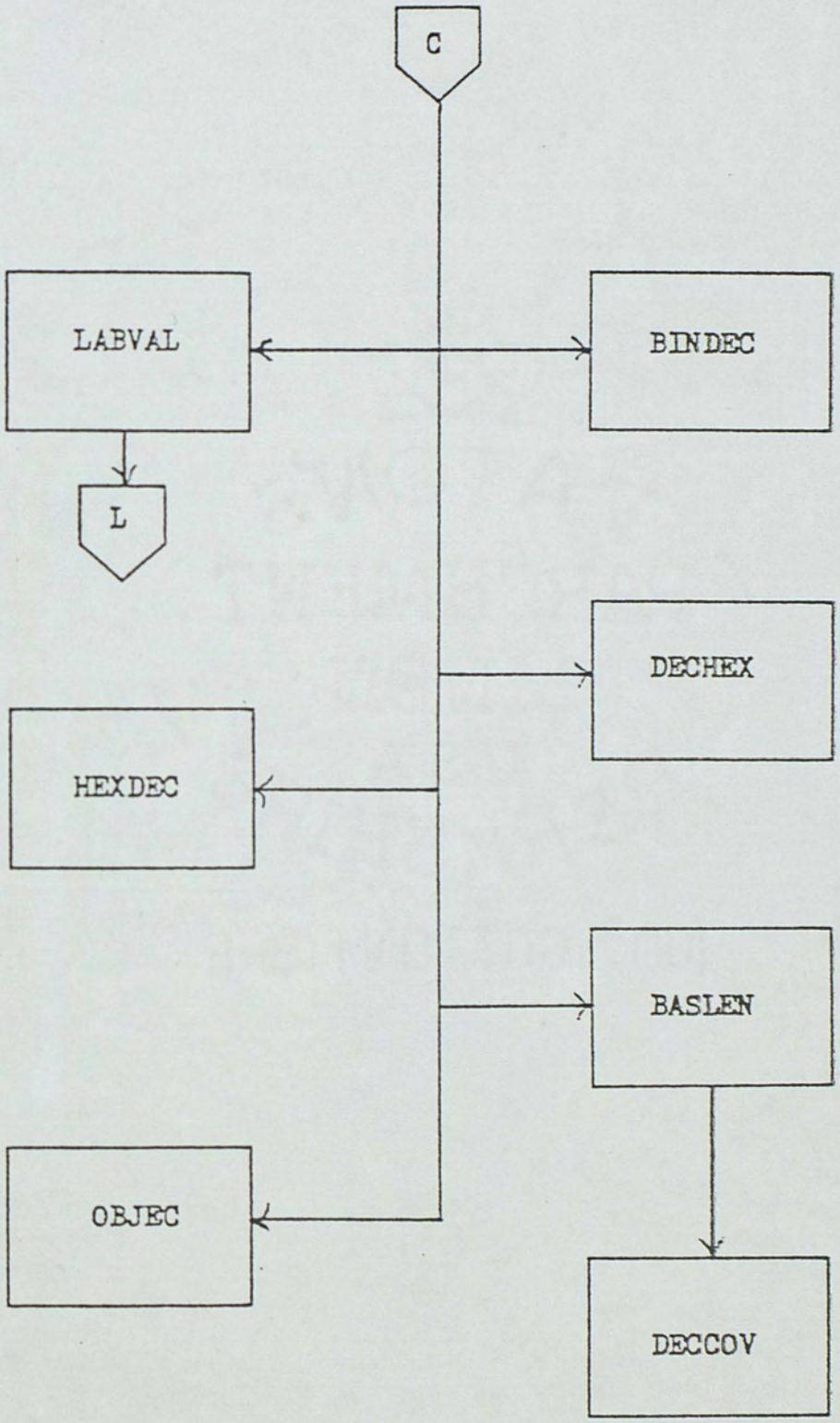

Fig. 5. EVOPAN block diagram

If a label was not found, the BASLEN subroutine is called to determine which number system is used, the length of the operand in HEX, and a count of the number of bytes occupied by the operand. The next step is to evaluate the operand based on the number system used. If it is octal, the OCTDEC subroutine is used; if binary, the BINDEC is used; if an ASCII character, EVOPAN finds the decimal value; and if decimal, EVOPAN converts the decimal value to HEX. If the operand is a HEX value, it is stored in HEXV. No matter which number system is used, the decimal value of the operand is stored in NU; and the HEX value is stored in HEXV.

Before control is returned from this subroutine, it checks to see if the operand contains an expression. It does this by checking the next position of the BUFFER array for a *, +, -, or /. If one is found, the operation is stored; and the previous steps are repeated to find the value of the next position of the operand. These steps are repeated until the complete expression has been evaluated.

PSEUDO

The two arguments of PSEUDO that have not been defined previously are NPASSF, which indicates either PASS 1 or PASS 2, and IPSEU, which indicates if a pseudo op was found.

The PSEUDO subroutine examines the BUFFER array for the following pseudo ops, which are depicted in Figure 6: ORG, RMB, NAM, FCB, EQU, SPC, FDB, and FCC. For PASS 1, the only pseudo ops which do not cause their respective subroutines to be called

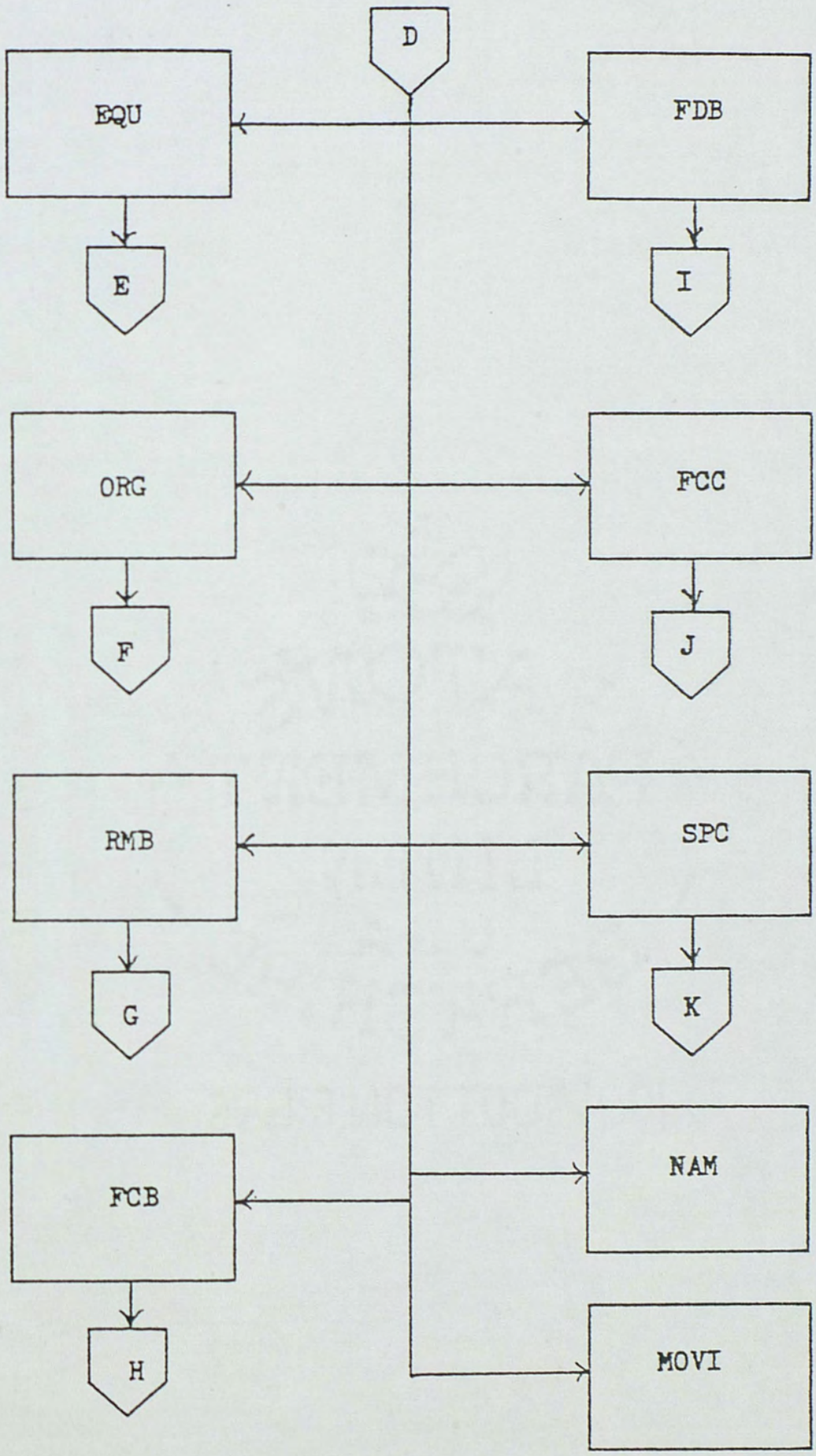

Fig. 6. PSEUDO block diagram

are: NAM, EQU, and SPC. During PASS 2, if any of the eight pseudo ops are found, the appropriate subroutine is called. In both PASS 1 and PASS 2 after the appropriate subroutine is called, the IPSEU flag is set and control returned to the calling program.

EQU

During PASS 1 the EQU subroutine evaluates the operand by calling EVOPAN, as depicted in Figure 7. It then tests the value of the operand to see if it is greater than 65535. If it is, an error message is generated. If not, SYMA is loaded with the HEX value of the operand.

During PASS 2 the variable SYM, which contains the label for EQU, is set as an argument in the SCHSYM subroutine to search the symbol table to find the HEX value for the operand. After this value is found, this line of the assembly listing is printed either to the CRT or printer.

ORG

For both PASS 1 and PASS 2, the ORG subroutine first calls EVOPAN, as depicted in Figure 8, to evaluate the operand. This value is then placed in LOCC and LOCD. Then LOCD is examined to see if it is greater than 65535. If it is, an error message is generated. During PASS 2, the assembly listing line is sent to the CRT or line printer. In addition, OBJEC is called to either write the previous line of object code or start another

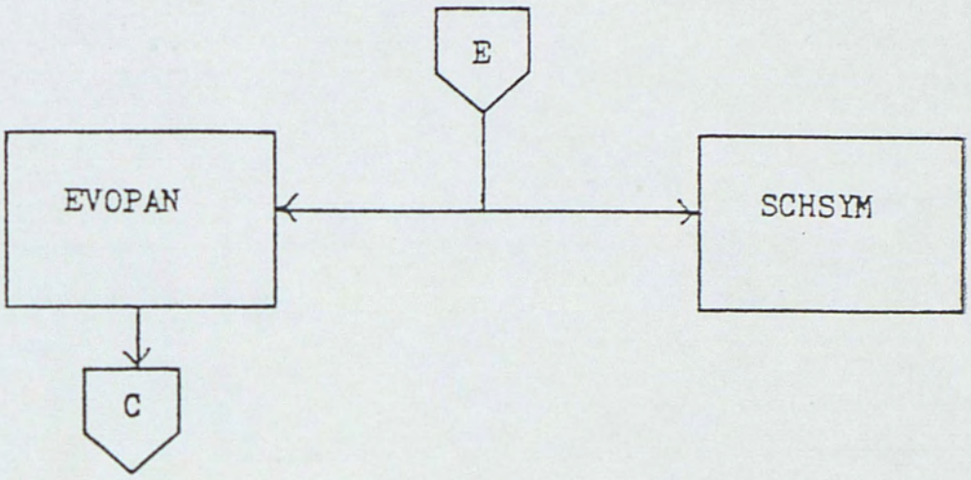

Fig. 7. EQU block diagram

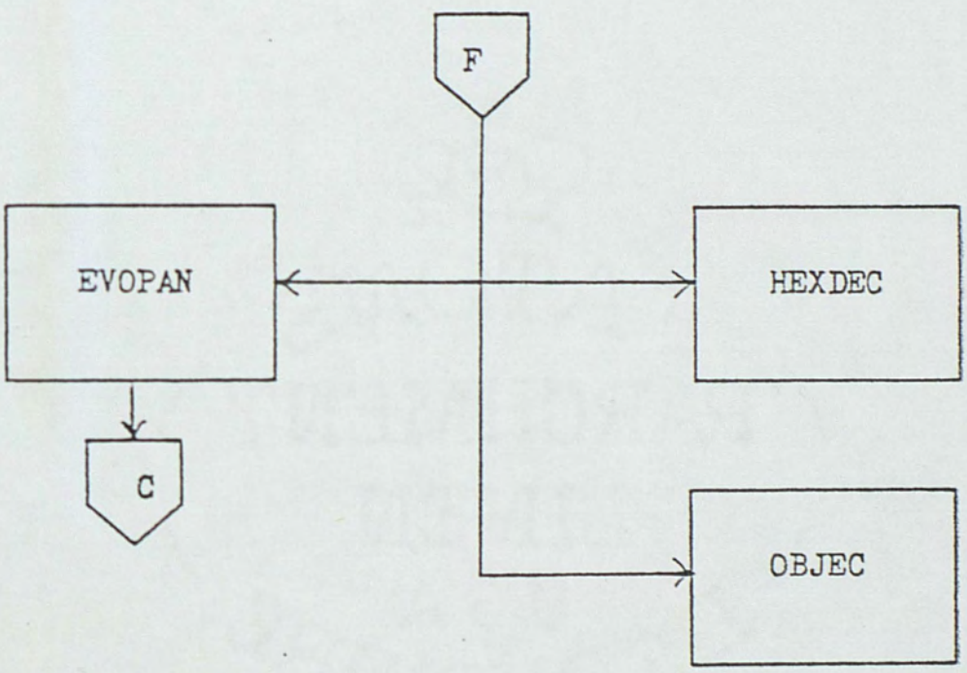

Fig. 8. ORG block diagram

line or initiate the object code line with the value found in the operand.

RMB

For PASS 1 EVOPAN is called, as depicted in Figure 9, to evaluate the operand. The value is checked to see if it is greater than 65535; and if it is, an error message is generated. If it isn't, the value is added to the location counter and control is returned to the calling subroutine.

During PASS 2 EVOPAN is called to evaluate the operand. This HEX value is then used to write the assembly line of code and its decimal value is used to repeatedly fill OBJT with zeros and to call OBJEC to insert the number of reserved bytes in the object file, as specified by the operand. Control is then returned to the calling subroutine.

FCB

During PASS 1 if FCB is called, it increments a counter which is set to 1 by 1 for each comma that is encountered in the operand. This value is then added to the location counter, and control is returned to the calling subroutine.

During PASS 2 each value appearing before a comma is evaluated by calling EVOPAN, as depicted in Figure 10. If the decimal value is greater than 255, an error message is generated. If not, the HEX value returned by EVOPAN is used to send the assembly listing line to the CRT or line printer. Then the

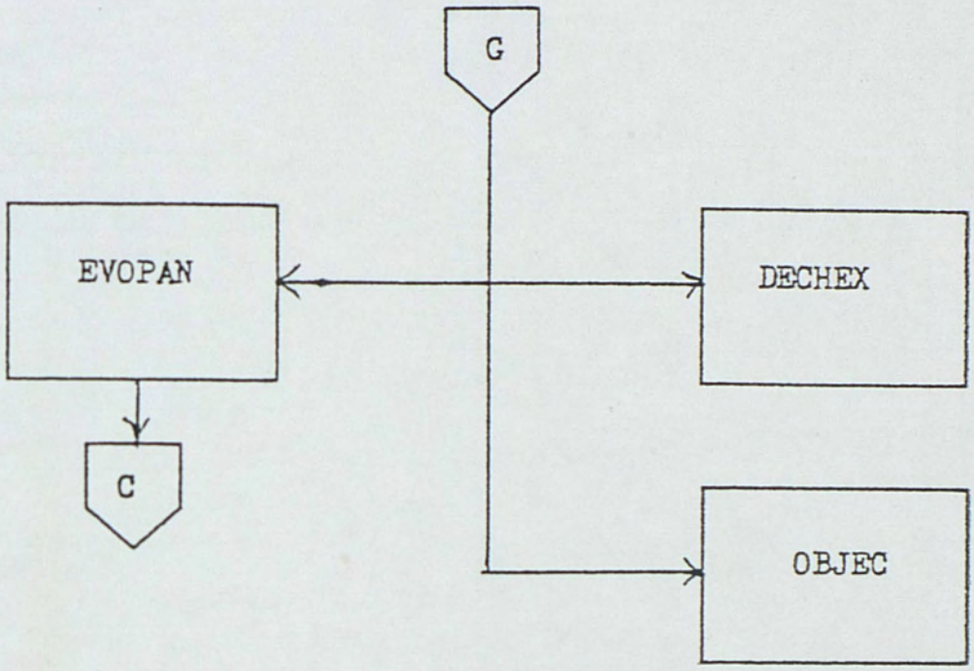

Fig. 9. RMB block diagram

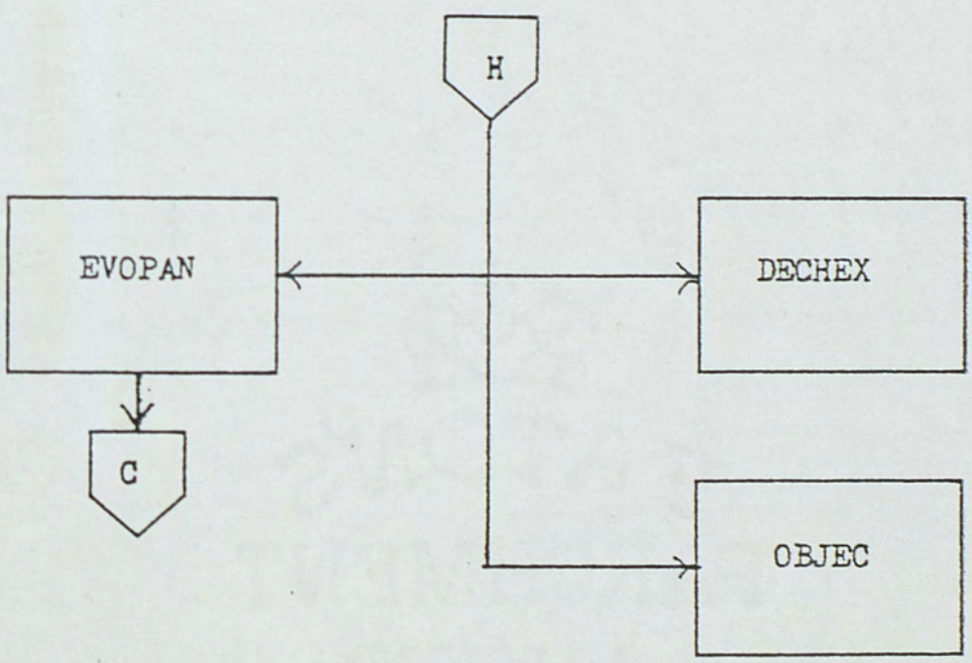

Fig. 10. FCB block diagram

OBJT array is loaded with the Hex code, and OBJEC is called to input this value to the object file. When a blank or tab is encountered, control is returned to the calling subroutine.

FDB

Figure 11 provides a block diagram for FDB. The FDB subroutine functions exactly like the FCB subroutine, except that each comma represents two bytes instead of one. This means that each value calculated by EVOPAN must be less than or equal to 65535, or an error message is generated.

FCC

During PASS 1 if a decimal value is used to indicate the length of the alphanumeric string, the decimal value is convertad from ASCII to a decimal value and used to increment the location counter. If the other method of defining a string is used, that being the first and last characters of the string being equal, then each position of the string is counted except the first and last position. This value is then used to increment the location counter. The value used to increment the location counter is tested to see if it is greater than 255. If it is, an error message is generated.

PASS 2 uses the same procedure as PASS 1 to determine the length of the string, but in addition it evaluates the ASCII characters to form their HEX values. It also writes the assembly listing lines and inserts the HEX values in OBJT and

calls OBJEC as depicted in Figure 12 to form the object code.

SPC

During PASS 2 the SPC subroutine provides a method for putting spaces between lines of the assembly listing. This is done by calling EVOPAN, as depicted in Figure 13, to evaluate the operand which represents the number of spaces to be left blank. This value is then used in a do loop to write blank lines on either the CRT or line printer.

LABVAL

The LABVAL subroutine places the label in the SYM array and calls SCHSYM as depicted in Figure 14. SCHSYM provides the HEX value for the label, and control is passed to the calling subroutine. If the label is longer than six bytes, an error message is generated.

OBJEC

OBJEC uses the argument LEN, which contains the number of bytes in OBJT, the other argument. OBJT is placed in the OBJECT file, and the location counter is incremented appropriately.

BASLEN

BASLEN checks the BUFFER array for a $ for HEX number, @ for octal number, % for binary number, ' for ASCII literal and * for the location counter. If none of these characters

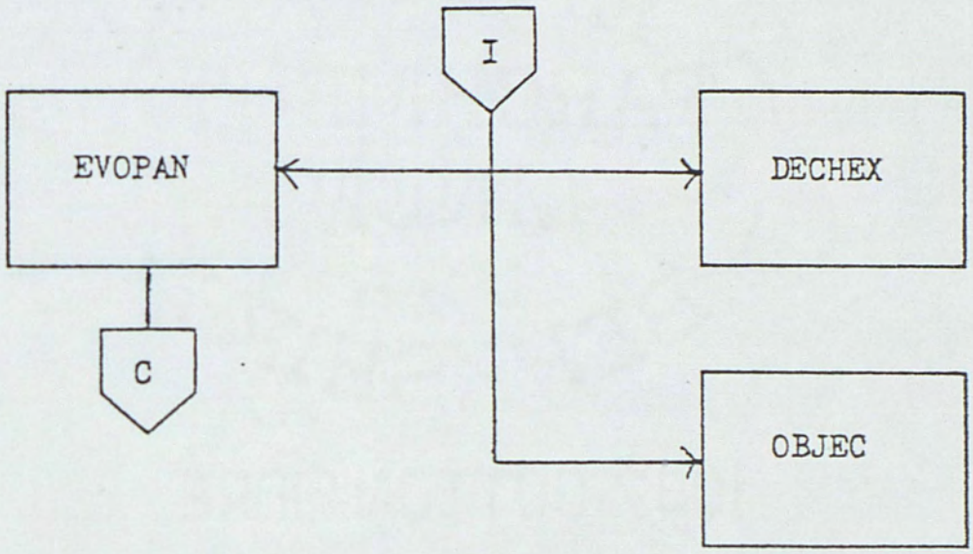

Fig. 11. FDB block diagram

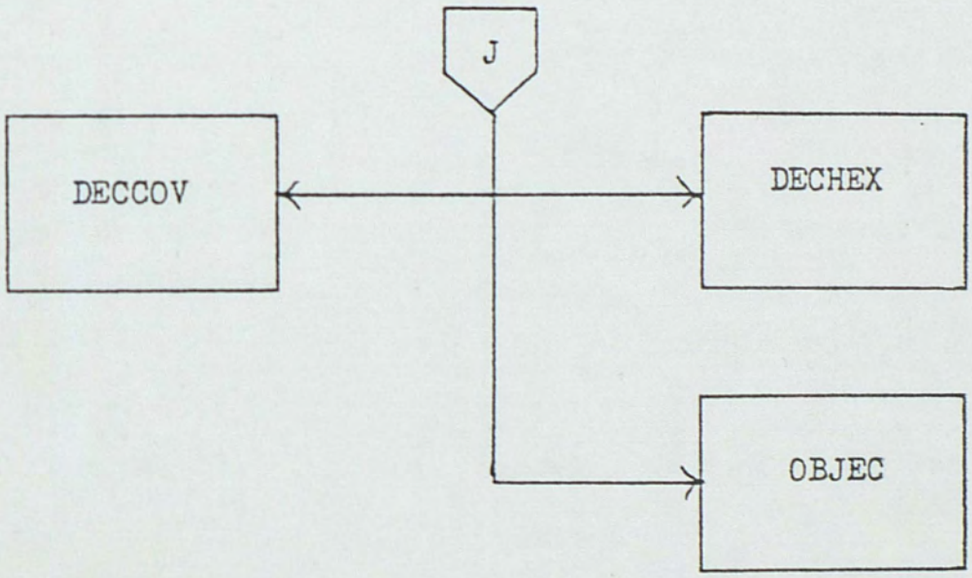

Fig. 12. FCC block diagram

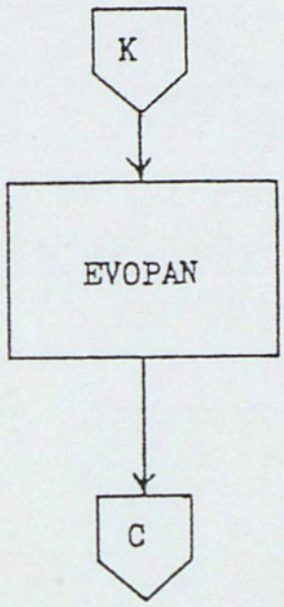

Fig. 13. SPC block diagram

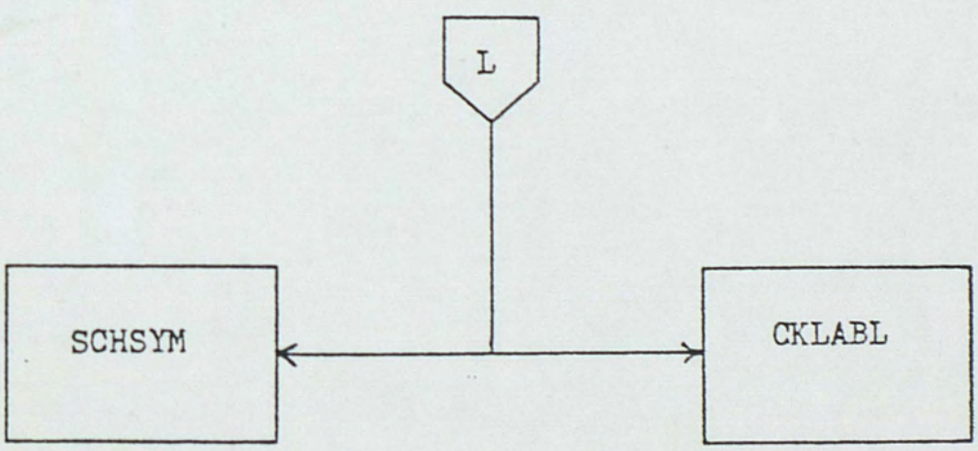

Fig. 14. LABVAL block diagram

are found, it is assumed that the BUFFER contains a decimal number. The NSYS is loaded with a value indicating which number system was used. In the case of a decimal number, DECCOV is called to convert the DEC byte array containing the ASCII numbers to a decimal number. In all cases the length in HEX and character positions is calculated and placed in LENH and LENB, respectively. Control is then returned to the calling subroutine.

SCHSYM

SCHSYM uses the IEFLAG flag to determine if the symbol table needs to be alphabetized. The SYM argument contains the symbol to be searched for; and HEXV contains the HEX value for the symbol, if it exists.

This subroutine uses a binary search to find a match with the first position of SYM and the first position of one of the symbols in the symbol table. If a match is found, a sequential search is started, first going forward. If no match is found, then the search moves backward. If no match is found, then an error message is generated. Once the search is finished and HEXV is loaded with the value for the symbol, control is returned to the calling subroutine.

ERRORS

The ERRORS subroutine writes the errors for a particular line of source code using the ERROR array, which contains the

error code and NERR, which contains the number of errors. After all errors have been written to either the CRT or line printer, control is passed back to the calling program.

PSYMT

The PSYMT writes the symbol table to either the line printer or CRT.

CKLABL

This checks the SYM argument to see if it is a valid label. If it is not, it generates an error message.

MOVI

MOVI moves the BUFFER pointer I until it finds an ASCII character, other than a blank or a TAB.

OPSCH

OPSCH searches the opcode table for the OPS argument. If a match is found, it returns the HEX value in the HEXOP argument. If no match is found, an error message is generated.

NAM

The NAM subroutine writes the NAM pseudo op to the CRT or line printer as part of the assembly listing.

HEXBIN

This converts the HEXC argument, which is a HEX value, to a binary number and loads it into the BIN argument.

TOCOMP

The TOCOMP does a two's complement operation on the BIN argument.

DECHEX

Converts the NUM argument to a HEX number and stores this in the HEXR argument. If NUM is greater than 65535, an error message is generated.

HEXDEC

Converts the HEX4 hex number to a decimal number and stores it in the NU argument. If HEX4 is not a HEX number, an error message is generated.

OCTDEC

This takes the argument OCTC, an octal number, and converts it to decimal, storing the value in the NU argument. If OCTC is not an octal number, an error message is generated.

BINDEC

The BINDEC converts the binary number contained in the BIN argument and loads it into NU argument. If BIN is not a binary number, an error message is generated.

DECCOV

The DECCOV takes the DEC argument, an ASCII representative of a decimal number, and converts it to a decimal number. It then stores it in the NU argument. If DEC is not a decimal number, an error message is generated.

CHAPTER III

CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

This report shows that a Motorola M6800 Assembler can be written for the PDP 11/34 to aid in assembling M6800 source code. Using an assembler alleviates the time-consuming burden of hand assembly. The assembler was written with ease of use in mind. It also was written to provide descriptive error messages. Once the program is running, it uses descriptive prompts for all inputs. Therefore, the user does not have to keep referring to a user's guide.

The M6800 Assembler written for the PDP 11/34 cannot run under TSX. Future work might include dividing the program into a PASS 1 assembler and a PASS 2 assembler. The user would first run PASS 1, which would create a symbol table and put it on a disk. Then the user would run the PASS 2 program, which would read the symbol table from disk and use it to create the assembly listing and object file. Dividing the assembler in this manner might allow the individual programs to run under TSX.

Another task for future work would be to develop a method for loading the object file which is on disk into the Motorola M6800 Microprocessor.

Both of these projects would provide needed tools for implementing M6800 source code.

APPENDIX A

M6800 ASSEMBLER USER'S GUIDE

FOR THE MOTOROLA M6800 MICROPROCESSOR

EXECUTING THE M6800 ASSEMBLER

The M6800 was programmed in Fortran to run on the PDP 11/34. To execute the M6800 Assembler, simply type RUN M6800 after the period prompt of the RT11 operating system. The following is a sample run with the underlined statement representing user inputs.

```
        ENTER LOGICAL UNIT NUMBER FOR LP OR TT > 7

        ENTER DEVICE AND FILENAME

        (i.e., DX1:NAME.FIL)  > DK1:M683.ASS

0000                ORG       0
0000     0002       INDEX     RMB     2
0000                ORG       $0C00
0C00     0001       PIAIAD    RMB     1
0C01     0001       PIAIAC    RMB     1
0C02     0001       PIAIBD    RMB     1
0C03     0001       PIAIBC    RMB     1
3F00                ORG       $3F00
3F00 01             TABLE     FCB     $1,$4F,$12,$6,$4C,$24,$60,$F,,$C
3F01 4F
3F02 12
3F03 06
3F04 4C
3F05 24
3F06 60
3F07 0F
3F00 00
3F09 0C
3F0A 30             FCB       $30,$30,$30,$30,$30,$30
3F0B 30
3F0C 30
3F0D 30
3F0E 30
3F0F 30
3FFE                ORG       $3FFE
```

```
3C00                ORG         $3C00
3C00 86 FF          START       LDA A       #$FF
3C05 86 04          LDA A       #%00000100
3C0D CE 3F00        LDX         #TABLE
3C15 84 0F          AND A       #%00001111
3C20 20 F0          BRA         LOOP
                    END

SYMBOL TABLE
INDEX   0000  LOOP   3C12  PIAIAC  0C01  PIAIAD  0C00  PIAIBC 0C03
PIAIBD  0C02  START  3C00  TABLE   3F00

YOUR OBJECT LISTING IS LOCATED IN DK1:M683.OBJ

STOP
```

.TYPE DK1:M683.OBJ

```
S1050000000000
S1070C000000000000
S1133F00014F12064C24600F000C303030303073
S1033FFE00
S1133C0086FFB80C028604B70C01B70C03CE3F004A
S1133C1100B60C00840F9701DE00A600B70C022046
S1033C2200
```

The input file DK1:M683.ASS is listed below:

```
          SPC       1
          ORG       0
INDEX     RMB       2
          ORG       $0C00
PIAIAD    RMB       1
PIAIAC    RMB       1
PIAIBD    RMB       1
PIAIBC    RMB       1
          ORG       $3F00
TABLE     FCB       $1,$4F,$12,$6,$4C,$24,$60,$F,,$C
          FCB       $30,$30,$30,$30,$30,$30
          ORG       $3FFE
          SPC       1
          ORG       $3C00
START     LDA A     #$FF
          STA A     PIAIBD
          LDA A     #%00000100
          STA A     PIAIAC
          STA A     PIAIBC
          LDX       #TABLE
          STX       INDEX
LOOP      LDA A     PIAIAD
          AND A     #%00001111
          STA A     INDEX+1
          LDX       INDEX
          LDA A     0,X
          STA A     PIAIBD
          BRA       LOOP
          END
```

The above example uses logical unit number 7 for the CRT.

The logical unit number for the line printer is 6.

APPENDIX B

OPCODE TABLE FOR THE

MOTOROLA M6800 MICROPROCESSOR

Table 1

Opcode Table

OPCODE	OPHEX	OPCODE	OPHEX	OPCODE	OPHEX
ABA H	1B	ADC A I	89	ADC A D	99
ADC A N	A9	ADC A E	B9	ADC B I	C9
ADC B D	D9	ADC B N	E9	ADC B E	F9
ADD A I	8B	ADD A D	9B	ADD A N	AB
ADD A E	BB	ADD B I	CB	ADD B D	DB
ADD B N	EB	ADD B E	FB	AND A I	84
AND A D	94	AND A N	A4	AND A E	B4
AND B I	C4	AND B D	D4	AND B N	E4
AND B E	F4	ASL N	68	ASL E	79
ASL A H	48	ASL B H	58	ASR N	67
ASR E	77	ASR A H	47	ASR B H	57
BCC R	24	BCS R	25	BEQ R	27
BGE R	2C	BGT R	2E	BHI R	22
BIT A I	85	BIT A D	95	BIT A N	A5
BIT A E	B5	BIT B I	C5	BIT B D	D5
BIT B N	E5	BIT B E	F5	BLE R	2F
BLS R	23	BLT R	2D	BMI R	2B
BNE R	26	BPL R	2A	BRA R	20
BSR R	8D	BVC R	28	BVS R	29
CBA H	11	CLC H	0C	CLI H	0E
CLR N	6F	CLR E	7F	CLR A H	4F
CLR B H	5F	CLV H	0A	CMP A I	81
CMP A D	91	CMP A N	A1	CMP A E	B1
CMP B I	C1	CMP B D	D1	CMP B N	E1
CMP B E	F1	COM N	63	COM E	73
COM A H	43	COM B H	53	CPX I	8C
CPX D	9C	CPX N	AC	CPX E	BC
DAA H	19	DEC N	6A	DEC E	7A
DEC A H	4A	DEC B H	5A	DES H	34
DEX H	09	EOR A I	88	EOR A D	98
EOR A N	A8	EOR A E	B8	EOR B I	C8
EOR B D	D8	EOR B N	E8	EOR B E	F8
INC N	6C	INC E	7C	INC A H	4C
INC B H	5C	INS H	31	INX H	08
JMP N	6E	JMP E	7E	JSR N	AD
JSR E	BD	LDA A I	86	LDA A D	96
LDA A N	A6	LDA A E	B6	LDA B I	C6

Table 1 (continued)

LDA	B	D	D6	LDA	B	N	E6	LDA	B	E	F6
LDS		I	8E	LDS		D	9E	LDS		N	AE
LDS		E	BE	LDX		I	CE	LDX		D	DE
LDX		N	EE	LDX		E	FE	LSR		N	64
LSR		E	74	LSR	A	H	44	LSR	B	H	54
NEG		N	60	NEG		E	70	NEG	A	H	40
NEG	B	H	50	NOP		H	01	ORA	A	I	8A
ORA	A	D	9A	ORA	A	N	AA	ORA	A	E	BA
ORA	B	I	CA	ORA	B	D	DA	ORA	B	N	EA
ORA	B	E	FA	PSH	A	H	36	PSH	B	H	37
PUL	A	H	32	PUL	B	H	33	ROL		N	69
ROL		E	79	ROL	A	H	49	ROL	B	H	59
ROR		N	66	ROR		E	76	ROR	A	H	46
ROR	B	H	56	RTI		H	3B	RTS		H	39
SBA		H	10	SBC	A	I	82	SBC	A	D	92
SBC	A	N	A2	SBC	A	E	B2	SBC	B	I	C2
SBC	B	D	D2	SBC	B	N	E2	SBC	B	E	F2
SEC		H	0D	SEC		H	0F	SEV		H	0B
STA	A	D	97	STA	A	N	A7	STA	A	E	B7
STA	B	D	D7	STA	B	N	E7	STA	B	E	F7
STS		D	9F	STS		N	AF	STS		E	BF
STX		D	DF	STX		N	EF	STX		E	FF
SUB	A	I	80	SUB	A	D	90	SUB	A	N	A0
SUB	A	E	B0	SUB	B	I	C0	SUB	B	D	D0
SUB	B	N	E0	SUB	B	E	F0	SWI		H	3F
TAB		H	16	TAP		H	06	TBA		H	17
TPA		H	07	TST		N	6D	TST		E	7D
TST	A	H	4D	TST	B	H	5D	TSX		H	30
TXS		H	35	WAI		H	3E				

* **

* Accumulator A or B or neither

** Addressing mode: I = immediate D = direct N = index
E = extended H = inherent R = relative

APPENDIX C

COMPUTER PROGRAM LISTING

FOR THE M6800 ASSEMBLER

```
MOTOROLA 6800 ASSEMBLER MAIN PROGRAM
REAL LOCD,NUMBER
INTEGER HEXB(15,4),BIN8(8),BIN16(16),ERROR(10)
BYTE FNAME(20),BUFFER(80),OPTAB(197,5),HEXT(15),OCTT(7)
1,HEXA(2),OCTN(2),SYMT(100,6),LOCC(4),SYMA(100,4),HEXV(4)
1,OBJF(42),OPHEX(197,2),LNAME(20),ONAME(20)
COMMON /C/HEXB/D/BUFFER,SYMA,SYMT
1/E/LOCC,LOCD/F/IFLAG/G/OPTAB,OPHEX
1/H/OBJF/I/NEND,NSTART/J/NORG/K/NUMBER,NSTOBJ
1/L/ERROR,NERR/M/KWRI
DATA NERR,ERROR/0,10*0/
DATA IFLAG,LNAME,ONAME,FNAME/1,20*' ',20*' ',20*' '/
DATA OBJF,NUMBER,NSTOBJ,NEND,NSTART/42*' ',0.,9,0,1/
DATA LOCC,LOCD,SYMA,SYMT,M/4*'0',0.,400*'0',600*' ',1/
DATA BIN8/1,1,1,1,0,0,0,0/
DATA HEXB/0,0,0,0,0,0,0,1,1,1
1,1,1,1,1,1,0,0,0,1,1,1,1
1,0,0,0,0,1,1,1,1,0,1,1,0
1,0,1,1,0,0,1,1,0,0,1,1,1
1,0,1,0,1,0,1,0,1,0,1,0,1
1,0,1/
DATA OPHEX/'1','8','9','A','B','C','D','E','F','8'
1,'9','A','B','C','D','E','F','8','9','A'
1,'B','C','D','E','F','6','7','4','5','6'
1,'7','4','5',6*'2','8','9','A','B','C','D','E','F',7*'2'
1,'8','2','2','1','0','0','6','7','4','5'
1,'0','8','9','A','B','C','D','E','F','6'
1,'7','4','5','8','9','A','B','1','6','7'
1,'4','5','3','0','8','9','A','B','C','D'
1,'E','F','6','7','4','5','3','0','6','7'
1,'A','B','8','9','A','B','C','D','E','F'
1,'8','9','A','B','C','D','E','F','6','7'
1,'4','5','6','7','4','5','0','8','9','A'
1,'B','C','D','E','F',4*'3','6','7','4','5'
1,'6','7','4','5','3','3','1','8','9','A'
1,'B','C','D','E','F','0','0','0','9','A'
1,'B','D','E','F','9','A','B','D'
1,'E','F','8','9','A','B','C','D','E','F'
1,'3','1','0','1','0','6','7','4','5',3*'3'
1,'B',8*'9',8*'B',8*'4','8','9','8','8',4*'7'
1,'4','5','7','C','E','2',8*'5','F','3','D'
```

```
1,'B','6','A','0','D','8','9','1','C','E'
1,4*'F','A',8*'1',4*'3',4*'C','9',4*'A','4'.
1,'9',8*'8',4*'C','1','8','E','E','D','D'
1,8*'6',8*'E',4*'4',4*'0','1',8*'A','6'
1,'7','2','3',4*'9',4*'6','B','9','0'
1,8*'2','D','F','B',6*'7',6*'F',8*'0'
1,'F','6','6','7','7',4*'D','0','5','E'/
DATA OPTAB/'A','A','A','A','A','A','A'
1,'A','A','A','A','A','A','A','A','A','A'
1,'A','A','A','A','A','A','A','A','A','A'
1,'A','A','A','A','A','A','B','B','B','B'
1,'B','B','B','B','B','B','B','B','B','B'
1,'B','B','B','B','B','B','B','B','B','B'
1,'C','C','C','C','C','C','C','C','C','C'
1,'C','C','C','C','C','C','C','C','C','C'
1,'C','C','C','C','D','D','D','D','D','D'
1,'D','E','E','E','E','E','E','E','E','I'
1,'I','I','I','I','I','J','J','J','J','L'
1,'L','L','L','L','L','L','L','L','L','L'
1,'L','L','L','L','L','L','L','L','L','N'
1,'N','N','N','N','O','O','O','O','O'
1,'O','O','O','P','P','P','P','R','R','R'
1,'R','R','R','R','R','R','R','S','S','S'
1,'S','S','S','S','S','S','S','S','S','S'
1,'S','S','S','S','S','S','S','S','S','S'
1,'S','S','S','S','S','S','S','S','S','S'
1,'T','T','T','T','T','T','T','T','T','T'
1,'W','B','D','D','D','D','D','D','D','D'
1,'D','D','D','D','D','D','D','D','N','N'
1,'N','N','N','N','N','N','S','S','S','S'
1,'S','S','S','S','C','C','E','G','G','H'
1,'I','I','I','I','I','I','I','I','L','L'
1,'L','M','N','P','R','S','V','V','B','L'
1,'L','L','L','L','L','L','M','M','M','M'
1,'M','M','M','M','O','O','O','O','P','P'
1,'P','P','A','E','E','E','E','E','E','O'
1,'O','O','O','O','O','O','O','N','N','N'
1,'N','N','N','M','M','S','S','D','D','D'
1,'D','D','D','D','D','D','D','D','D','D'
1,'D','D','D','S','S','S','S','E','E','E'
1,'E','O','R','R','R','R','R','R','R','R'
1,'S','S','U','U','O','O','O','O','O','O'
1,'O','O','T','T','B','B','B','B','B'
```

```
1,'B','B','B','B','E','E','E','T','T','T'
1,'T','T','T','T','T','T','T','T','T','U'
1,'U','U','U','U','U','U','U','W','A','A'
1,'B','P','S','S','S','S','S','X','A','A'
1,'C','C','C','C','C','C','C','C','D','D'
1,'D','D','D','D','D','D','D','D','D','D'
1,'D','D','D','D','L','L','L','L','R','R'
1,'R','R','C','S','Q','E','T','I','T','T'
1,'T','T','T','T','T','T','E','S','T','I'
1,'E','L','A','R','C','S','A','C','I','R'
1,'R','R','R','V','P','P','P','P','P','P'
1,'P','P','M','M','M','M','X','X','X','X'
1,'A','C','C','C','C','S','X','R','R','R'
1,'R','R','R','R','R','C','C','C','C','S'
1,'X','P','P','R','R','A','A','A','A','A'
1,'A','A','A','S','S','S','S','X','X','X'
1,'X','R','R','R','R','G','G','G','G','P','A'
1,'A','A','A','A','A','A','A','H','H','L'
1,'L','L','L','L','L','R','R','R','R','I'
1,'S','A','C','C','C','C','C','C','C','C'
1,'C','C','V','A','A','A','A','A','A','S'
1,'S','S','X','X','X','B','B','B','B','B'
1,'B','B','B','I','B','P','A','A','T','T'
1,'T','T','X','S','I',' ','A','A','A','A'
1,'B','B','B','B','A','A','A','A','B','B'
1,'B','B','A','A','A','A','B','B','B','B'
1,' ',' ','A','B',' ',' ','A','B',' ',' '
1,' ',' ',' ',' ','A','A','A','A','B','B'
1,'B','B',' ',' ',' ',' ',' ',' ',' ',' '
1,' ',' ',' ',' ',' ',' ',' ','A','B',' '
1,'A','A','A','A','B','B','B','B',' ',' '
1,'A','B',' ',' ',' ',' ',' ',' ',' ','A'
1,'B',' ',' ','A','A','A','A','B','B','B'
1,'B',' ',' ','A','B',' ',' ',' ',' ',' '
1,' ','A','A','A','A','B','B','B','B',' '
1,' ',' ',' ',' ',' ',' ',' ',' ',' ','A'
1,'B',' ',' ','A','B',' ','A','A','A','A'
1,'B','B','B','B','A','B','A','B',' ',' '
1,'A','B',' ',' ','A','B',' ',' ',' ','A'
1,'A','A','A','B','B','B','B',' ',' ',' '
1,'A','A','A','B','B','B',' ',' ',' ',' '
```

```
     1,' ',' ','A','A','A','A','B','B','B','B'
     1,' ',' ',' ',' ',' ',' ',' ','A','B',' '
     1,' ',' ','H','I','D','N','E','I','D','N'
     1,'E','I','D','N','E','I','D','N','E','I'
     1,'D','N','E','I','D','N','E','N','E','H'
     1,'H','N','E','H','H','R','R','R','R','R'
     1,'R','I','D','N','E','I','D','N','E','R'
     1,'R','R','R','R','R','R','R','R','R','H'
     1,'H','H','N','E','H','H','H','I','D','N'
     1,'E','I','D','N','E','N','E','H','H','I'
     1,'D','N','E','H','N','E','H','H','H','H'
     1,'I','D','N','E','I','D','N','E','N','E'
     1,'H','H','H','H','N','E','N','E','I','D'
     1,'N','E','I','D','N','E','I','D','N','E'
     1,'I','D','N','E','N','E','H','H','N','E'
     1,'H','H','H','I','D','N','E','I','D','N'
     1,'E','H','H','H','H','N','E','H','H','N'
     1,'E','H','H','H','H','H','I','D','N','E','I','D'
     1,'N','E','H','H','H','D','N','E','D','N'
     1,'E','D','N','E','D','N','E','I','D','N'
     1,'E','I','D','N','E','H','H','H','H','H'
     1,'N','E','H','H','H','H','H'/
      WRITE(7,120)
120   FORMAT(1X,'ENTER LOGICAL UNIT NUMBER FOR',1X,
     1'LP OR TT > ',$)
      READ(5,125) KWRI
125   FORMAT(I2)
      LERR=0
      LERR2=0
      WRITE(7,10)
10    FORMAT(1X,'ENTER DEVICE AND FILENAME'
     1,/,1X,'(I.E. DX1:NAME.FIL) >',$)
      READ(5,20) (FNAME(I),I=1,20)
20    FORMAT(20A1)
      DO 90 L=1,20
      ONAME(L)=FNAME(L)
      LNAME(L)=FNAME(L)
      IF(FNAME(L).EQ.'.') GO TO 100
90    CONTINUE
100   LNAME(L+1)='L'
      LNAME(L+2)='S'
      LNAME(L+3)='T'
      ONAME(L+1)='O'
      ONAME(L+2)='B'
```

```
      ONAME(L+3)='J'
      OPEN(UNIT=2,NAME=ONAME,TYPE='NEW',RECORDSIZE=55,
      1INITIALSIZE=20)
5     NPASSF=1
15    OPEN(UNIT=3,NAME=FNAME,TYPE='OLD',RECORDSIZE=80)
50    READ(3,30) (BUFFER(I),I=1,80)
30    FORMAT(80A1)
      IF(BUFFER(I+1).EQ.'*'.AND.NPASSF.EQ.2) CALL NAM
      IF(BUFFER(I+1).EQ.'*') GO TO 50
      I=1
      IF(NPASSF.EQ.2) GO TO 25
      CALL PASS1(M)
      IF(NERR.EQ.0) GO TO 35
      WRITE(7,30) (BUFFER(L),L=1,80)
      CALL ERRORS
      LERR=1
35    IF(NEND.EQ.1) GO TO 110
      GO TO 50
110   CLOSE(UNIT=3)
      LOCD=0
      DO 45 L=1,4
      LOCC(L)='0'
45    CONTINUE
      NEND=0
      NPASSF=2
      OPEN(UNIT=3,NAME=FNAME,TYPE='OLD',RECORDSIZE=80)
      GO TO 50
25    IF(LERR.EQ.1) GO TO 75
      CALL PASS2(M)
      IF(NERR.EQ.0) GO TO 65
      CALL ERRORS
      LERR2=1
65    IF(NEND.EQ.1) GO TO 115
      GO TO 50
115   CALL OBJEC(0,OBJF)
      CALL NAM
      CALL SCHSYM(0,M,SYM,HEXV)
      CALL PSYMT(M)
      IF(LERR2.EQ.0) GO TO 95
      GO TO 75
95    WRITE(7,105) ONAME
105   FORMAT(1X,'YOUR OBJECT LISTING IS LOCATED IN',
      11X,20A1)
75    CLOSE(UNIT=2)
      CLOSE(UNIT=3)
      END
```

```
      SUBROUTINE PASS1(M)
      REAL NU,LOCD
      INTEGER ERROR(10)
      BYTE SYMT(100,6),LOCC(4),BUFFER(80),SYMA(100,4)
     1,HEXV(4),SYM(6),HEXOP(2),OPS(5)
      COMMON /D/BUFFER,SYMA,SYMT/E/LOCC,LOCD
     1/L/ERROR,NERR/I/NEND
      DO 5 LOOP=1,6
      SYM(LOOP)=' '
5     CONTINUE
      NPASSF=1
      J=1
      I=1
      IF(BUFFER(I).EQ.' '.AND.BUFFER(I+1).EQ.' ')
     1GO TO 10
      IF(BUFFER(I).EQ.'        ')
     1 GO TO 10
20    I=I+1
      IF(BUFFER(I).EQ.' ') GO TO 25
      IF(BUFFER(I).EQ.'        ') GO TO 25
      IF(J.EQ.7) GO TO 95
      SYM(J)=BUFFER(I)
      SYMT(M,J)=BUFFER(I)
      J=J+1
      GO TO 20
25    CALL CKLABL(SYM)
      IF(NERR.EQ.0) GO TO 45
      IF(ERROR(NERR).EQ.3) GO TO 105
45    IF(M.EQ.1) GO TO 35
      CALL SCHSYM(1,M,SYM,HEXV)
      IF(NERR.EQ.0) GO TO 125
      IF(ERROR(NERR).NE.2) GO TO 125
      ERROR(NERR)=0
      NERR=NERR-1
35    CALL MOVI(I)
      I=I+1
      IF(BUFFER(I).EQ.'E'.AND.BUFFER(I+1).EQ.'R'.AND.
     1BUFFER(I+2).EQ.'U') GO TO 30
      DO 80 JJ=1,4
      SYMA(M,JJ)=LOCC(JJ)
80    CONTINUE
```

```
      M=M+1
      I=I-1
      GO TO 10
30    I=I+3
      CALL EQU(NPASSF,I,M,SYM)
      RETURN
10    CALL MOVI(I)
      I=I+1
      IF(BUFFER(I).EQ.'E'.AND.BUFFER(I+1).EQ.'N'.AND.
     1BUFFER(I+2).EQ.'D') NEND=1
      IF(BUFFER(I).EQ.'E'.AND.BUFFER(I+1).EQ.'N'.AND.
     1BUFFER(I+2).EQ.'D') RETURN
      CALL PSEUDO(NPASSF,I,IPSEU,M)
      IF(IPSEU.EQ.1) GO TO 90
      K=I
      IF(BUFFER(I).NE.'B') GO TO 40
      IF(BUFFER(I+1).EQ.'I') GO TO 40
      LOCD=LOCD+2
      CALL DECHEX(LOCD,LOCC,4)
      RETURN
40    DO 170 IO=1,3
      OPS(IO)=BUFFER(I)
      I=I+1
170   CONTINUE
      OPS(4)=' '
      IF(BUFFER(I+1).EQ.'A'.AND.BUFFER(I+2).EQ.' ')
     1 OPS(4)='A'
      IF(BUFFER(I+1).EQ.'A'.AND.BUFFER(I+2).EQ.' ')
     1 I=I+2
      IF(BUFFER(I+1).EQ.'A'.AND.BUFFER(I+2).EQ.'  ')
     1 OPS(4)='A'
      IF(BUFFER(I+1).EQ.'A'.AND.BUFFER(I+2).EQ.'  ')
     1 I=I+2
      IF(BUFFER(I+1).EQ.'B'.AND.BUFFER(I+2).EQ.' ')
     1 OPS(4)='B'
      IF(BUFFER(I+1).EQ.'B'.AND.BUFFER(I+2).EQ.' ')
     1 I=I+2
      IF(BUFFER(I+1).EQ.'B'.AND.BUFFER(I+2).EQ.'  ')
     1 OPS(4)='B'
      IF(BUFFER(I+1).EQ.'B'.AND.BUFFER(I+2).EQ.'  ')
     1 I=I+2
      OPS(5)='H'
      CALL OPSCH(OPS,HEXOP)
```

```
      IF(NERR.EQ.0) GO TO 65
      IF(ERROR(NERR).NE.5) GO TO 65
      ERROR(NERR)=0
      NERR=NERR-1
      CALL MOVI(I)
      IF(I.EQ.80) RETURN
      I=I+1
      IF(BUFFER(I).NE.'$') GO TO 50
      IF(BUFFER(K).EQ.'C'.AND.BUFFER(K+1).EQ.'P'
     1.AND.BUFFER(K+2).EQ.'X'.OR.BUFFER(K).EQ.'L'
     1.AND.BUFFER(K+1).EQ.'D'.AND.BUFFER(K+2).EQ.'S'
     1.OR.BUFFER(K).EQ.'L'.AND.BUFFER(K+1).EQ.'D'
     1.AND.BUFFER(K+2).EQ.'X') GO TO 100
      LOCD=LOCD+2
      CALL DECHEX(LOCD,LOCC,4)
      RETURN
100   LOCD=LOCD+3
      CALL DECHEX(LOCD,LOCC,4)
      RETURN
50    I=I-1
      IF(BUFFER(I+1).EQ.'X'.AND.BUFFER(I+2).EQ.' ')
     1GO TO 110
      IF(BUFFER(I+1).EQ.'X'.AND.BUFFER(I+2).EQ.'      ')
     1GO TO 110
      IF(BUFFER(I+1).EQ.','.AND.BUFFER(I+2).EQ.'X'
     1.AND.BUFFER(I+3).EQ.' ') GO TO 110
      IF(BUFFER(I+1).EQ.','.AND.BUFFER(I+2).EQ.'X'
     1.AND.BUFFER(I+3).EQ.'  ') GO TO 110
120   DO 60 KK=1,20
      IF(BUFFER(I+KK).NE.',') GO TO 60
      IF(BUFFER(I+KK+1).NE.'X') GO TO 60
110   LOCD=LOCD+2
      CALL DECHEX(LOCD,LOCC,4)
      RETURN
60    CONTINUE
      GO TO 70
65    LOCD=LOCD+1
      CALL DECHEX(LOCD,LOCC,4)
      RETURN
70    CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
      IF(NU.LE.255.) GO TO 75
      LOCD=LOCD+3
      GO TO 85
75    LOCD=LOCD+2
85    CALL DECHEX(LOCD,LOCC,4)
90    RETURN
125   NERR=NERR+1
      ERROR(NERR)=22
      GO TO 105
```

```
95      NERR=NERR+1
        ERROR(NERR)=3
105     DO 115 KK=1,6
        SYMT(M,KK)=' '
115     CONTINUE
        RETURN
        END
```

```
      SUBROUTINE PASS2(M)
      REAL NU,NDIS,LOCD
      INTEGER BIN(16),ERROR(10)
      BYTE LOCC(4),BUFFER(80),HEXV(4),SYM(6),OPS(5)
     1,HEXOP(2),OBJT(6)
      COMMON /D/BUFFER/E/LOCC,LOCD/I/NEND/L/ERROR,NERR
     1/M/KWRI
      DATA OBJT/6*'0'/
      DATA SYM/6*' '/
      NPASSF=2
      J=1
      I=1
      IF(BUFFER(I).EQ.' '.AND.BUFFER(I+1).EQ.' ')
     1 GO TO 10
      IF(BUFFER(I).EQ.'        ') GO TO 10
20    I=I+1
      IF(BUFFER(I).EQ.' ') GO TO 25
      IF(BUFFER(I).EQ.'        ') GO TO 25
      SYM(J)=BUFFER(I)
      J=J+1
      GO TO 20
25    CALL MOVI(I)
      I=I+1
      IF(BUFFER(I).EQ.'E'.AND.BUFFER(I+1).EQ.'Q'.AND.
     1BUFFER(I+2).EQ.'U') GO TO 30
      I=I-1
      GO TO 10
30    I=I+3
      CALL EQU(NPASSF,I,M,SYM)
      RETURN
10    CALL MOVI(I)
      I=I+1
      IF(BUFFER(I).EQ.'E'.AND.BUFFER(I+1).EQ.'N'.AND.
     1BUFFER(I+2).EQ.'D') NEND=1
      IF(BUFFER(I).EQ.'E'.AND.BUFFER(I+1).EQ.'N'.AND.
     1BUFFER(I+2).EQ.'D') RETURN
      CALL PSEUDO(NPASSF,I,IPSEU,M)
      IF(IPSEU.EQ.1) GO TO 90
      K=I
      IF(BUFFER(I).NE.'B') GO TO 40
      IF(BUFFER(I+1).EQ.'I') GO TO 40
```

```
      DO 130 IO=1,3
      OPS(IO)=BUFFER(I)
      I=I+1
130   CONTINUE
      OPS(4)=' '
      OPS(5)='R'
      CALL OPSCH(OPS,HEXOP)
      I=I-1
      CALL MOVI(I)
      CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
      NDIS=NU-(LOCD+2)
      IF(NDIS.LT.-128..OR.NDIS.GT.127.) GO TO 105
      IF(NDIS.LT.0) GO TO 140
      CALL DECHEX(NDIS,HEXV,4)
      GO TO 150
140   NDIS=ABS(NDIS)
      CALL DECHEX(NDIS,HEXV,4)
      CALL HEXBIN(HEXV,BIN)
      CALL TOCOMP(BIN)
      CALL BINDEC(BIN,16,NU)
      CALL DECHEX(NU,HEXV,4)
      GO TO 150
105   HEXV(3)='0'
      HEXV(4)='0'
      NERR=NERR+1
      ERROR(NERR)=12
150   WRITE(KWRI,160) (LOCC(L),L=1,4),(HEXOP(L),L=1,2),
      1(HEXV(L),L=3,4),(BUFFER(L),L=1,80)
160   FORMAT(1X,4A1,1X,2A1,1X,2A1,5X,80A1)
      OBJT(1)=HEXOP(1)
      OBJT(2)=HEXOP(2)
      OBJT(3)=HEXV(3)
      OBJT(4)=HEXV(4)
      LEN=2
      CALL OBJEC(LEN,OBJT)
      RETURN
40    DO 170 IO=1,3
      OPS(IO)=BUFFER(I)
      I=I+1
170   CONTINUE
      OPS(4)=' '
      IF(BUFFER(I+1).EQ.'A'.AND.BUFFER(I+2).EQ.' ')
      1 OPS(4)='A'
```

```
      IF(BUFFER(I+1).EQ.'A'.AND.BUFFER(I+2).EQ.' ')
     1 I=I+2
      IF(BUFFER(I+1).EQ.'A'.AND.BUFFER(I+2).EQ.'       ')
     1 OPS(4)='A'
      IF(BUFFER(I+1).EQ.'A'.AND.BUFFER(I+2).EQ.'       ')
     1 I=I+2
      IF(BUFFER(I+1).EQ.'B'.AND.BUFFER(I+2).EQ.' ')
     1 OPS(4)='B'
      IF(BUFFER(I+1).EQ.'B'.AND.BUFFER(I+2).EQ.' ')
     1 I=I+2
      IF(BUFFER(I+1).EQ.'B'.AND.BUFFER(I+2).EQ.'       ')
     1 OPS(4)='B'
      IF(BUFFER(I+1).EQ.'B'.AND.BUFFER(I+2).EQ.'       ')
     1 I=I+2
      OPS(5)='H'
      CALL OPSCH(OPS,HEXOP)
      IF(NERR.EQ.0) GO TO 65
      IF(ERROR(NERR).NE.5) GO TO 65
      ERROR(NERR)=0
      NERR=NERR-1
      CALL MOVI(I)
      IF(I.EQ.80) GO TO 65
      I=I+1
      IF(BUFFER(I).NE.'*') GO TO 50
      IF(BUFFER(K).EQ.'C'.AND.BUFFER(K+1).EQ.'P'
     1.AND.BUFFER(K+2).EQ.'X'.OR.BUFFER(K).EQ.'L'
     1.AND.BUFFER(K+1).EQ.'D'.AND.BUFFER(K+2).EQ.'S'
     1.OR.BUFFER(K).EQ.'L'.AND.BUFFER(K+1).EQ.'D'
     1.AND.BUFFER(K+2).EQ.'X') GO TO 100
      OPS(5)='I'
      CALL OPSCH(OPS,HEXOP)
D     WRITE(7,200) I,BUFFER(I)
D200  FORMAT(1X,I3,A1)
      CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
      WRITE(KWRI,160) (LOCC(L),L=1,4),(HEXOP(L),L=1,2),
     1(HEXV(L),L=3,4),(BUFFER(L),L=1,80)
      OBJT(1)=HEXOP(1)
      OBJT(2)=HEXOP(2)
      OBJT(3)=HEXV(3)
      OBJT(4)=HEXV(4)
      LEN=2
      CALL OBJEC(LEN,OBJT)
5     IF(NU.GE.0..AND.NU.LE.255.) RETURN
```

```
      NERR=NERR+1
      ERROR(NERR)=13
      RETURN
100   OPS(5)='I'
      CALL OPSCH(OPS,HEXOP)
      CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
      WRITE(KWRI,180) (LOCC(L),L=1,4),(HEXOP(L),L=1,2),
     1(HEXV(L),L=1,4),(BUFFER(L),L=1,80)
180   FORMAT(1X,4A1,1X,2A1,1X,4A1,3X,80A1)
      OBJT(1)=HEXOP(1)
      OBJT(2)=HEXOP(2)
      OBJT(3)=HEXV(1)
      OBJT(4)=HEXV(2)
      OBJT(5)=HEXV(3)
      OBJT(6)=HEXV(4)
      LEN=3
      CALL OBJEC(LEN,OBJT)
15    IF(NU.GE.0..AND.NU.LE.65535.) RETURN
      NERR=NERR+1
      ERROR(NERR)=14
      RETURN
50    I=I-1
      IF(BUFFER(I+1).EQ.'X'.AND.BUFFER(I+2).EQ.' ')
     1GO TO 115
      IF(BUFFER(I+1).EQ.','.AND.BUFFER(I+2).EQ.'X'
     1.AND.BUFFER(I+3).EQ.' ') GO TO 115
      IF(BUFFER(I+1).EQ.'X'.AND.BUFFER(I+2).EQ.'      ')
     1GO TO 115
      IF(BUFFER(I+1).EQ.','.AND.BUFFER(I+2).EQ.'X'.AND.
     1BUFFER(I+3).EQ.'        ') GO TO 115
      GO TO 120
115   HEXV(3)='0'
      HEXV(4)='0'
      GO TO 110
120   DO 60 KK=1,20
      IF(BUFFER(I+KK).NE.',') GO TO 60
      IF(BUFFER(I+KK+1).NE.'X') GO TO 60
      CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
110   OPS(5)='N'
      CALL OPSCH(OPS,HEXOP)
      WRITE(7,160) (LOCC(L),L=1,4),(HEXOP(L),L=1,2),
     1(HEXV(L),L=3,4),(BUFFER(L),L=1,80)
      OBJT(1)=HEXOP(1)
      OBJT(2)=HEXOP(2)
      OBJT(3)=HEXV(3)
      OBJT(4)=HEXV(4)
      LEN=2
      CALL OBJEC(LEN,OBJT)
```

```
      GO TO 5
60    CONTINUE
      GO TO 70
65    WRITE(7,190) (LOCC(L),L=1,4),(HEXOP(L),L=1,2),
      1(BUFFER(L),L=1,80)
190   FORMAT(1X,4A1,1X,2A1,8X,80A1)
      OBJT(1)=HEXOP(1)
      OBJT(2)=HEXOP(2)
      LEN=1
      CALL OBJEC(LEN,OBJT)
      RETURN
70    CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
      IF(NU.LE.255) GO TO 75
      OPS(5)='E'
      CALL OPSCH(OPS,HEXOP)
      WRITE(7,180) (LOCC(L),L=1,4),(HEXOP(L),L=1,2),
      1(HEXV(L),L=1,4),(BUFFER(L),L=1,80)
      OBJT(1)=HEXOP(1)
      OBJT(2)=HEXOP(2)
      OBJT(3)=HEXV(1)
      OBJT(4)=HEXV(2)
      OBJT(5)=HEXV(3)
      OBJT(6)=HEXV(4)
      LEN=3
      CALL OBJEC(LEN,OBJT)
      GO TO 15
75    OPS(5)='D'
      CALL OPSCH(OPS,HEXOP)
      WRITE(7,160) (LOCC(L),L=1,4),(HEXOP(L),L=1,2),
      1(HEXV(L),L=3,4),(BUFFER(L),L=1,80)
      OBJT(1)=HEXOP(1)
      OBJT(2)=HEXOP(2)
      OBJT(3)=HEXV(3)
      OBJT(4)=HEXV(4)
      LEN=2
      CALL OBJEC(LEN,OBJT)
      GO TO 5
90    RETURN
      END
```

```
      SUBROUTINE EVOPAN(I,LENH,NU,NSYS,HEXV,M)
      REAL NU,LOCD,NUM,NUS
      INTEGER BIN16(16),ERROR(10)
      BYTE HEXV(4),BUFFER(80),OCT6(6),TEMP(2)
     1,LOCC(4)
      COMMON /D/BUFFER/E/LOCC,LOCD/L/ERROR,NERR
      IFLG=0
195   DO 5 JJ=1,6
      OCT6(JJ)='0'
5     CONTINUE
      DO 15 JJ=1,16
      BIN16(JJ)=0
15    CONTINUE
      DO 155 JJ=1,4
      HEXV(JJ)='0'
155   CONTINUE
      IF(BUFFER(I+1).GE.'A'.AND.BUFFER(I+1).LE.'Z')
     1 GO TO 70
      I=I+1
      CALL BASLEN(I,LENH,NU,LENB,NSYS)
      GO TO (10,20,30,40,50,65),NSYS
10    KNT=5-LENH
      DO 60 LL=KNT,4
      I=I+1
      HEXV(LL)=BUFFER(I)
60    CONTINUE
      CALL HEXDEC(HEXV,NU,4)
      GO TO 110
20    KNT=7-LENB
      DO 80 LL=KNT,6
      I=I+1
      OCT6(LL)=BUFFER(I)
80    CONTINUE
      CALL OCTDEC(OCT6,NU,6)
      GO TO 50
30    KNT=17-LENB
      DO 90 LL=KNT,16
      I=I+1
      BIN16(LL)=BUFFER(I)-48
90    CONTINUE
```

```
      CALL BINDEC(BIN16,16,NU)
      GO TO 50
40    I=I+1
      KNT=LENH
      NALF=BUFFER(I)
      IF(NALF.LT.32.OR.NALF.GT.95) GO TO 25
      CALL DECHEX(NALF,HEXV,4)
      TEMP(1)=HEXV(3)
      TEMP(2)=HEXV(4)
      KNT=KNT-1
      IF(KNT.EQ.0) GO TO 45
      I=I+1
      NALF=BUFFER(I)
      IF(NALF.LT.32.OR.NALF.GT.95) GO TO 25
      CALL DECHEX(NALF,HEXV,4)
      HEXV(1)=TEMP(1)
      HEXV(2)=TEMP(2)
45    CALL HEXDEC(HEXV,NU,4)
      GO TO 110
50    CALL DECHEX(NU,HEXV,4)
      GO TO 110
65    NU=LOCD
      DO 75 LL=1,4
      HEXV(LL)=LOCC(LL)
75    CONTINUE
      GO TO 110
70    CALL LABVAL(I,HEXV,M)
      CALL HEXDEC(HEXV,NU,4)
      I=I-1
      NSYS=6
110   I=I+1
      IF(IFLG.EQ.1) GO TO 120
      IF(BUFFER(I).NE.' '.AND.BUFFER(I).NE.','.AND.
     1BUFFER(I).NE.' ') GO TO 120
      RETURN
120   IF(IFLG.EQ.1) GO TO 130
      NUS=NU
      IF(BUFFER(I).EQ.'*') NSIGN=1
      IF(BUFFER(I).EQ.'+') NSIGN=2
      IF(BUFFER(I).EQ.'-') NSIGN=3
      IF(BUFFER(I).EQ.'/') NSIGN=4
      IFLG=1
      GO TO 195
```

```
130     CALL HEXDEC(HEXV,NUM,4)
        GO TO (140,150,160,170) ,NSIGN
140     NUS=NUM*NUS
        GO TO 190
150     NUS=NUM+NUS
        GO TO 190
160     NUS=NUS-NUM
        GO TO 190
170     NUS=NUS/NUM
190     CONTINUE
        IF(BUFFER(I).EQ.'*') NSIGN=1
        IF(BUFFER(I).EQ.'+') NSIGN=2
        IF(BUFFER(I).EQ.'-') NSIGN=3
        IF(BUFFER(I).EQ.'/') NSIGN=4
        IF(BUFFER(I).NE.' '.AND.BUFFER(I).NE.','.AND.
       1BUFFER(I).NE.' ') GO TO 195
180     NU=NUS
        CALL DECHEX(NUS,HEXV,4)
        RETURN
25      NERR=NERR+1
        ERROR(NERR)=11
        NU=0
        DO 55 LL=1,4
        HEXV(LL)='0'
55      CONTINUE
        RETURN
        END
```

```
      SUBROUTINE PSEUDO(NPASSF,I,IPSEU,M)
      BYTE BUFFER(80)
      COMMON /D/BUFFER
D     WRITE(7,200)
D 200 FORMAT(1X,'PSEUDO')
D     WRITE(7,210) (BUFFER(II),II=I,5)
D 210 FORMAT(1X,3A1)
D     WRITE(7,220) M
D 220 FORMAT(1X,'M=',I5)
      IF(BUFFER(I).EQ.'O'.AND.BUFFER(I+1).EQ.'R'.AND.
     1BUFFER(I+2).EQ.'G') GO TO 40
      IF(BUFFER(I).EQ.'R'.AND.BUFFER(I+1).EQ.'M'
     1.AND.BUFFER(I+2).EQ.'B') GO TO 50
      IF(BUFFER(I).EQ.'N'.AND.BUFFER(I+1).EQ.'A'.AND.
     1BUFFER(I+2).EQ.'M') GO TO 60
      IF(BUFFER(I).EQ.'F'.AND.BUFFER(I+1).EQ.'C'.AND.
     1BUFFER(I+2).EQ.'B') GO TO 70
      IF(BUFFER(I).EQ.'E'.AND.BUFFER(I+1).EQ.'Q'.AND.
     1BUFFER(I+2).EQ.'U') GO TO 80
      IF(BUFFER(I).EQ.'S'.AND.BUFFER(I+1).EQ.'P'.AND.
     1BUFFER(I+2).EQ.'C') GO TO 90
      IF(BUFFER(I).EQ.'F'.AND.BUFFER(I+1).EQ.'D'.AND.
     1BUFFER(I+2).EQ.'B') GO TO 100
      IF(BUFFER(I).EQ.'F'.AND.BUFFER(I+1).EQ.'C'.AND.
     1BUFFER(I+2).EQ.'C') GO TO 110
      IPSEU=0
      RETURN
 40   I=I+3
      CALL MOVI(I)
      CALL ORG(NPASSF,I,M)
      IPSEU=1
      RETURN
 50   I=I+3
      CALL MOVI(I)
      CALL RMB(M,NPASSF,I)
      IPSEU=1
      RETURN
 60   I=I+3
      CALL MOVI(I)
      IF(NPASSF.EQ.2) CALL NAM
      IPSEU=1
      RETURN
```

```
70      I=I+3
        CALL MOVI(I)
        CALL FCB(NPASSF,I,M)
        IPSEU=1
        RETURN
80      I=I+3
        CALL MOVI(I)
        IF(NPASSF.EQ.2) CALL EQU(NPASSF,I,M)
        IPSEU=1
        RETURN
90      I=I+3
        CALL MOVI(I)
        IF(NPASSF.EQ.2) CALL SPC(I,M)
        IPSEU=1
        RETURN
100     I=I+3
        CALL MOVI(I)
        CALL FDB(NPASSF,I,M)
        IPSEU=1
        RETURN
110     I=I+3
        CALL MOVI(I)
        CALL FCC(NPASSF,I)
        IPSEU=1
        RETURN
        END
```

```
      SUBROUTINE EQU(NPASSF,I,M,SYM)
      INTEGER ERROR(10)
      REAL NU
      BYTE HEXV(4),BUFFER(80),SYMA(100,4),SYM(6)
      COMMON /D/BUFFER,SYMA/L/ERROR,NERR/M/KWRI
      IF(NPASSF.EQ.2) GO TO 20
      CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
      IF(NU.GT.65535.) GO TO 40
      DO 10 LL=1,4
      SYMA(M,LL)=HEXV(LL)
10    CONTINUE
      M=M+1
      RETURN
20    CALL SCHSYM(1,M,SYM,HEXV)
      WRITE(KWRI,30) (HEXV(K),K=1,4),(BUFFER(K),K=1,80)
30    FORMAT(5X,4A1,7X,80A1)
      RETURN
40    DO 50 LL=1,4
      SYMA(M,LL)='0'
50    CONTINUE
      M=M+1
      NERR=NERR+1
      ERROR(NERR)=14
      RETURN
      END
```

```
      SUBROUTINE ORG(NPASSF,I,M)
      REAL LOCD,NU
      INTEGER ERROR(10)
      BYTE HEXV(4),LOCC(4),BUFFER(80)
     1,OBJT(6)
      COMMON /D/BUFFER/E/LOCC,LOCD/J/NORG/L/ERROR,NERR
     1/M/KWRI
D     WRITE(7,200)
D200  FORMAT(1X,'ORG')
      CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
      DO 10 LL=1,4
      LOCC(LL)=HEXV(LL)
 10   CONTINUE
      CALL HEXDEC(LOCC,NU,4)
      LOCD=NU
      IF(NU.GT.65535.) GO TO 40
      IF(NPASSF.EQ.2) GO TO 20
      RETURN
 20   WRITE(KWRI,30) (LOCC(K),K=1,4),(BUFFER(K),K=1,80)
 30   FORMAT(1X,4A1,11X,80A1)
      NORG=1
      CALL OBJEC(0,OBJT)
      RETURN
 40   LOCD=0
      DO 50 LL=1,4
      LOCC(LL)='0'
 50   CONTINUE
      NERR=NERR+1
      ERROR(NERR)=14
      RETURN
      END
```

```
      SUBROUTINE RMB(M,NPASSF,I)
      REAL LOCD,NU
      INTEGER ERROR(10)
      BYTE BUFFER(80),LOCC(4),HEXV(4),OBJT(6)
      COMMON /D/BUFFER/E/LOCC,LOCD/L/ERROR,NERR/M/KWRI
      DATA OBJT/6*'0'/
D     WRITE(7,200)
D200  FORMAT(1X,'RMB')
      CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
      IF(NPASSF.EQ.1) GO TO 30
      WRITE(KWRI,10) (LOCC(K),K=1,4),(HEXV(K),K=1,4)
     1,(BUFFER(K),K=1,80)
 10   FORMAT(1X,4A1,4X,4A1,3X,80A1)
      K=NU
      DO 40 L=1,K
      OBJT(1)='0'
      OBJT(2)='0'
D     WRITE(7,210) OBJT(1),OBJT(2)
D210  FORMAT(1X,2A1)
      LEN=1
      CALL OBJEC(LEN,OBJT)
 40   CONTINUE
      RETURN
 30   IF(NU.GT.65535.) GO TO 50
      LOCD=LOCD+NU
      CALL DECHEX(LOCD,LOCC,4)
D     WRITE(7,210) NU,LOCD
      RETURN
 50   NERR=NERR+1
      ERROR(NERR)=14
      RETURN
      END
```

```
      SUBROUTINE FCB(NPASSF,I,M)
      REAL NU,LOCD,J
      INTEGER ERROR(10)
      BYTE BUFFER(80),LOCC(4),HEXV(4)
     1,OBJT(6)
      COMMON /D/BUFFER/E/LOCC,LOCD,/L/ERROR,NERR
     1/M/KWRI
      DATA OBJT/6*'0'/
      WRITE(7,200)
  200 FORMAT(1X,'FCB')
      IF(NPASSF.EQ.2) GO TO 30
      I=I+1
      J=1
   20 IF(BUFFER(I).EQ.' ') GO TO 10
      IF(BUFFER(I).EQ.',') J=J+1
      I=I+1
      GO TO 20
   10 LOCD=LOCD+J
      CALL DECHEX(LOCD,LOCC,4)
      RETURN
   30 J=I+1
      IF(BUFFER(J).EQ.',') GO TO 40
      GO TO 35
   80 IF(BUFFER(I).EQ.','.AND.BUFFER(I+1).EQ.',') GO TO 45
      IF(BUFFER(I).EQ.','.AND.BUFFER(I+1).EQ.' ') GO TO 60
      IF(BUFFER(I).EQ.','.AND.BUFFER(I+1).EQ.'          ')
     1 GO TO 60
      IF(BUFFER(I).EQ.' '.OR.BUFFER(I).EQ.'    ') RETURN
      CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
      IF(NU.GT.255.) GO TO 90
      WRITE(KWRI,70) (LOCC(K),K=1,4),(HEXV(K),K=3,4)
   70 FORMAT(1X,4A1,1X,2A1)
      OBJT(1)=HEXV(3)
      OBJT(2)=HEXV(4)
      LEN=1
      CALL OBJEC(LEN,OBJT)
      GO TO 80
   90 NERR=NERR+1
      ERROR(NERR)=13
      GO TO 80
```

```
35      CALL EVOPAN(I,LEN,NU,NSYS,HEXV,M)
        IF(NU.GT.255.) GO TO 95
        WRITE(KWRI,75) (LOCC(K),K=1,4),(HEXV(K),K=3,4),
        1(BUFFER(K),K=1,80)
75      FORMAT(1X,4A1,1X,2A1,8X,80A1)
        OBJT(1)=HEXV(3)
        OBJT(2)=HEXV(4)
        LEN=1
        CALL OBJEC(LEN,OBJT)
        GO TO 80
95      NERR=NERR+1
        ERROR(NERR)=13
        WRITE(KWRI,85) (LOCC(K),K=1,4),(BUFFER(K),K=1,80)
85      FORMAT(1X,4A1,11X,80A1)
        GO TO 80
45      WRITE(KWRI,100) (LOCC(K),K=1,4)
        I=I+1
        OBJT(1)='0'
        OBJT(2)='0'
        LEN=1
        CALL OBJEC(LEN,OBJT)
        GO TO 80
40      WRITE(KWRI,100) (LOCC(K),K=1,4),(BUFFER(K),K=1,80)
100     FORMAT(1X,4A1,1X,'00',8X,80A1)
        OBJT(1)='0'
        OBJT(2)='0'
        LEN=1
        CALL OBJEC(LEN,OBJT)
        I=I+1
        GO TO 80
60      WRITE(KWRI,100) (LOCC(K),K=1,4),(BUFFER(K),K=1,80)
        OBJT(1)='0'
        OBJT(2)='0'
        LEN=1
        CALL OBJEC(LEN,OBJT)
        RETURN
        END
```

```
      SUBROUTINE FDB(NPASSF,I,M)
      REAL NU,LOCD,J
      INTEGER ERROR(10)
      BYTE BUFFER(80),LOCC(4),HEXV(4)
     1,OBJT(6)
      COMMON /D/BUFFER/E/LOCC,LOCD/L/ERROR,NERR
     1/M/KWRI
      DATA OBJT/6*'0'/
D     WRITE(7,200)
D200  FORMAT(1X,'FDB')
      IF(NPASSF.EQ.2) GO TO 30
      I=I+1
      J=2
 20   IF(BUFFER(I).EQ.' ') GO TO 10
      IF(BUFFER(I).EQ.',') J=J+2
      I=I+1
      GO TO 20
 10   LOCD=LOCD+J
      CALL DECHEX(LOCD,LOCC,4)
      RETURN
 30   J=I+1
      IF(BUFFER(J).EQ.',') GO TO 40
      GO TO 35
 80   IF(BUFFER(I).EQ.','.AND.BUFFER(I+1).EQ.',')
     1GO TO 45
      IF(BUFFER(I).EQ.','.AND.BUFFER(I+1).EQ.' ')
     1GO TO 60
      IF(BUFFER(I).EQ.','.AND.BUFFER(I+1).EQ.
     1'      ') GO TO 60
      IF(BUFFER(I).EQ.' '.OR.BUFFER(I).EQ.'    ')
     1RETURN
      CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
      IF(NU.GT.65535.) GO TO 90
      WRITE(KWRI,70) (LOCC(K),K=1,4),(HEXV(K),K=1,4)
 70   FORMAT(1X,4A1,1X,4A1)
      OBJT(1)=HEXV(1)
      OBJT(2)=HEXV(2)
      OBJT(3)=HEXV(3)
      OBJT(4)=HEXV(4)
      LEN=2
      CALL OBJEC(LEN,OBJT)
      GO TO 80
```

```
90      NERR=NERR+1
        ERROR(NERR)=14
        GO TO 80
35      CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
        IF(NU.GT.65535.) GO TO 95
        WRITE(KWRI,75) (LOCC(K),K=1,4),(HEXV(K),K=1,4),
        1(BUFFER(K),K=1,80)
75      FORMAT(1X,4A1,1X,4A1,6X,80A1)
        OBJT(1)=HEXV(1)
        OBJT(2)=HEXV(2)
        OBJT(3)=HEXV(3)
        OBJT(4)=HEXV(4)
        LEN=2
        CALL OBJEC(LEN,OBJT)
        GO TO 80
95      NERR=NERR+1
        ERROR(NERR)=14
        WRITE(KWRI,85)(LOCC(K),K=1,4),(BUFFER(K),K=1,80)
85      FORMAT(1X,4A1,11X,80A1)
        GO TO 80
45      WRITE(KWRI,100) (LOCC(K),K=1,4)
        I=I+1
        OBJT(1)='0'
        OBJT(2)='0'
        OBJT(3)='0'
        OBJT(4)='0'
        LEN=2
        CALL OBJEC(LEN,OBJT)
        GO TO 80
40      WRITE(KWRI,100) (LOCC(K),K=1,4),(BUFFER(K),K=1,80)
100     FORMAT(1X,4A1,1X,'0000',6X,80A1)
        OBJT(1)='0'
        OBJT(2)='0'
        OBJT(3)='0'
        OBJT(4)='0'
        LEN=2
        CALL OBJEC(LEN,OBJT)
        I=I+1
        GO TO 80
60      WRITE(KWRI,100) (LOCC(K),K=1,4),(BUFFER(K),K=1,80)
        OBJT(1)='0'
        OBJT(2)='0'
        OBJT(3)='0'
        OBJT(4)='0'
        LEN=2
        CALL OBJEC(LEN,OBJT)
        RETURN
        END
```

```
      SUBROUTINE FCC(NPASSF,I)
      INTEGER ERROR(10),WFLG
      REAL LOCD,NU,NUM
      BYTE BUFFER(80),DEC(5),LOCC(4),HEXV(4)
     1,OBJT(6)
      COMMON /D/BUFFER/E/LOCC,LOCD/L/ERROR,NERR
     1/M/KWRI
      WFLG=0
      DO 65 J=1,6
      OBJT(J)='0'
65    CONTINUE
      DO 75 J=1,5
      DEC(J)='0'
75    CONTINUE
      I=I+1
      J=I
      IF(BUFFER(I).LT.'0'.OR.BUFFER(I).GT.'9')
     1GO TO 50
10    I=I+1
      IF(BUFFER(I).GE.'0'.AND.BUFFER(I).LE.'9')
     1GO TO 10
      IF(BUFFER(I).NE.',') GO TO 50
      MM=I-J
      M=6-MM
      DO 30 LL=M,5
      DEC(LL)=BUFFER(J)
      J=J+1
30    CONTINUE
      CALL DECCOV(DEC,NU)
      IF(NPASSF.EQ.1) GO TO 80
      K=NU
      DO 60 L=1,K
      I=I+1
      NUM=BUFFER(I)
      CALL DECHEX(NUM,HEXV,4)
      IF(WFLG.EQ.1) GO TO 5
      WRITE(KWRI,100) (LOCC(K),K=1,4),(HEXV(K),K=3,4)
     1,(BUFFER(K),K=1,80)
      WFLG=1
      GO TO 15
5     WRITE(KWRI,100) (LOCC(K),K=1,4),(HEXV(K),K=3,4)
100   FORMAT(1X,4A1,1X,2A1,8X,80A1)
15    OBJT(1)=HEXV(3)
      OBJT(2)=HEXV(4)
      LEN=1
      CALL OBJEC(LEN,OBJT)
60    CONTINUE
      RETURN
```

```
50      I=I+1
        IF(BUFFER(I).NE.BUFFER(J)) GO TO 50
        J=J+1
        K=I-J
        NU=K
        IF(NPASSF.EQ.1) GO TO 80
        DO 70 L=1,K
        NUM=BUFFER(J)
        CALL DECHEX(NUM,HEXV,4)
        J=J+1
        IF(WFLG.EQ.1) GO TO 25
        WRITE(KWRI,100) (LOCC(K),K=1,4),(HEXV(K),K=3,4),
        1(BUFFER(K),K=1,80)
        WFLG=1
        GO TO 35
25      WRITE(KWRI,100) (LOCC(K),K=1,4),(HEXV(K),K=3,4)
35      OBJT(1)=HEXV(3)
        OBJT(2)=HEXV(4)
        LEN=1
        CALL OBJEC(LEN,OBJT)
70      CONTINUE
        RETURN
80      IF(NU.GT.255.) GO TO 40
        LOCD=LOCD+NU
        CALL DECHEX(LOCD,LOCC,4)
        RETURN
40      NERR=NERR+1
        ERROR(NERR)=20
        RETURN
        END
```

```
      SUBROUTINE SPC(I,M)
      REAL NU
      BYTE BUFFER(80),HEXV(4)
      COMMON /D/BUFFER/M/KWRI
      CALL EVOPAN(I,LENH,NU,NSYS,HEXV,M)
      K=NU
      DO 20 J=1,K
      WRITE(KWRI,10)
10    FORMAT(1X)
20    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE LABVAL(I,HEXV,M)
      INTEGER ERROR(10)
      BYTE BUFFER(80),SYM(6),HEXV(4)
      COMMON /D/BUFFER/L/ERROR,NERR
D     WRITE(7,200)
D200  FORMAT(1X,'LABVAL')
      DO 50 JJ=1,6
      SYM(JJ)=' '
 50   CONTINUE
      NUM=0
      KK=1
 40   I=I+1
      IF(BUFFER(I).EQ.' '.OR.BUFFER(I).EQ.'+'.OR.
     1BUFFER(I).EQ.'-'.OR.BUFFER(I).EQ.'*'.OR.
     1BUFFER(I).EQ.'/') GO TO 20
      IF(KK.EQ.7) GO TO 30
      SYM(KK)=BUFFER(I)
D     WRITE(7,210) BUFFER(I),SYM(KK),KK,I
D210  FORMAT(1X,2A1,2I3)
      KK=KK+1
      GO TO 40
 20   CONTINUE
D     WRITE(7,230) (SYM(L),L=1,6)
D230  FORMAT(1X,'SYM ',6A1)
      CALL CKLABL(SYM)
      IF(NERR.EQ.0) GO TO 25
      IF(ERROR(NERR).EQ.3) GO TO 30
 25   CONTINUE
D     WRITE(7,220) M
D220  FORMAT(1X,'M=',I5)
      CALL SCHSYM(1,M,SYM,HEXV)
      RETURN
 30   I=I+1
      IF(BUFFER(I).EQ.' '.OR.BUFFER(I).EQ.'+'.OR.
     1BUFFER(I).EQ.'-'.OR.BUFFER(I).EQ.'*'.OR.
     1BUFFER(I).EQ.'/') GO TO 70
      GO TO 30
 70   NERR=NERR+1
      ERROR(NERR)=3
      DO 60 L=1,4
      HEXV(L)='0'
 60   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE OBJEC(LEN,OBJT)
      REAL LOCD,NUMBER,NU
      BYTE OBJF(42),OBJT(6),LOCC(4),HEXV(4)
      COMMON /E/LOCC,LOCD,/H/OBJF/I/NEND,NSTART/J/NORG
     1/K/NUMBER,NSTOBJ
      IF(NORG.EQ.1.AND.OBJF(1).EQ.' ') GO TO 30
      IF(NSTART.EQ.1) GO TO 30
      IF(NORG.EQ.1) GO TO 70
      IF(NEND.EQ.1) GO TO 70
45    I=1
      NDOBJ=42
40    OBJF(NSTOBJ)=OBJT(I)
      OBJF(NSTOBJ+1)=OBJT(I+1)
      HEXV(3)=OBJT(I)
      HEXV(4)=OBJT(I+1)
      HEXV(1)='0'
      HEXV(2)='0'
      NSTOBJ=NSTOBJ+2
      I=I+2
      LEN=LEN-1
      CALL HEXDEC(HEXV,NU,4)
      NUMBER=NUMBER+NU
      LOCD=LOCD+1
      CALL DECHEX(LOCD,LOCC,4)
      IF(NSTOBJ.GT.NDOBJ) GO TO 10
      IF(LEN.EQ.0) RETURN
      GO TO 40
10    OBJF(3)='1'
      OBJF(4)='3'
      CALL DECHEX(NUMBER,HEXV,4)
      OBJF(41)=HEXV(3)
      OBJF(42)=HEXV(4)
      WRITE(2,50) (OBJF(L),L=1,42)
50    FORMAT(1X,42A1)
      NSTOBJ=9
      NUMBER=0
      DO 60 L=1,4
      OBJF(L+4)=LOCC(L)
60    CONTINUE
```

```
      OBJF(1)='S'
      OBJF(2)='1'
      IF(LEN.EQ.0) RETURN
      GO TO 40
30    DO 20 L=1,4
      OBJF(L+4)=LOCC(L)
20    CONTINUE
      OBJF(1)='S'
      OBJF(2)='1'
      NSTART=0
      NORG=0
      IF(LEN.NE.0) GO TO 45
      RETURN
70    NU=NSTOBJ
      NU=(NU-2)/2
      CALL DECHEX(NU,HEXV,4)
      OBJF(3)=HEXV(3)
      OBJF(4)=HEXV(4)
      CALL DECHEX(NUMBER,HEXV,4)
      OBJF(NSTOBJ)=HEXV(3)
      OBJF(NSTOBJ+1)=HEXV(4)
      WRITE(2,50) (OBJF(L),L=1,NSTOBJ+1)
      IF(NEND.EQ.1) RETURN
      NDOBJ=42
      NORG=0
      NSTOBJ=9
      NUMBER=0
      DO 80 L=1,4
      OBJF(L+4)=LOCC(L)
80    CONTINUE
      OBJF(1)='S'
      OBJF(2)='1'
      RETURN
      END
```

```
      SUBROUTINE BASLEN(II,LENH,NU,LENB,NSYS)
      REAL NU
      INTEGER ERROR(10)
      BYTE BUFFER(80),DEC(5)
      COMMON /D/BUFFER/L/ERROR,NERR
      DO 100 IDEC=1,5
      DEC(IDEC)='0'
100   CONTINUE
      LENH=-1
      LENB=-1
      NU=0
      K=II
      IF(BUFFER(II).EQ.'$') GO TO 10
      IF(BUFFER(II).EQ.'@') GO TO 20
      IF(BUFFER(II).EQ.'%') GO TO 30
      III=BUFFER(II)
      IF(III.EQ.39) GO TO 40
      IF(BUFFER(II).EQ.'*') NSYS=6
      IF(BUFFER(II).EQ.'*') RETURN
      LENH=0
      NSYS=5
50    LENH=LENH+1
      K=K+1
      IF(BUFFER(K).NE.' '.AND.BUFFER(K).NE.'+'.AND.
     1BUFFER(K).NE.'-'.AND.BUFFER(K).NE.'*'.AND.
     1BUFFER(K).NE.'/'.AND.BUFFER(K).NE.','.AND.
     1BUFFER(K).NE.' ') GO TO 50
      K=II
      IF(LENH.LE.5) GO TO 55
      NERR=NERR+1
      ERROR(NERR)=14
      LL=LENH-5
      LENH=5
      II=II+LL
55    M=6-LENH
      DO 60 LL=M,5
      DEC(LL)=BUFFER(K)
      K=K+1
60    CONTINUE
      CALL DECCOV(DEC,NU)
      IF(NU.LT.255.) GO TO 70
      LENH=4
      GO TO 80
```

```
70      LENH=2
        GO TO 80
10      NSYS=1
90      LENH=LENH+1
        K=K+1
        IF(BUFFER(K).NE.' '.AND.BUFFER(K).NE.'+'.AND.
       1BUFFER(K).NE.'-'.AND.BUFFER(K).NE.'*'.AND.
       1BUFFER(K).NE.'/'.AND.BUFFER(K).NE.','.AND.
       1BUFFER(K).NE.' ') GO TO 90
        IF(LENH.LE.4) GO TO 80
        NERR=NERR+1
        ERROR(NERR)=14
        LL=LENH-4
        LENH=4
        II=II+LL
        GO TO 80
20      LENB=LENB+1
        K=K+1
        IF(BUFFER(K).NE.' '.AND.BUFFER(K).NE.'+'.AND.
       1BUFFER(K).NE.'-'.AND.BUFFER(K).NE.'*'.AND.
       1BUFFER(K).NE.'/'.AND.BUFFER(K).NE.','.AND.
       1BUFFER(K).NE.' ') GO TO 20
        IF(LENB.LE.6) GO TO 5
        NERR=NERR+1
        ERROR(NERR)=14
        LL=LENB-6
        LENB=6
        II=II+LL
5       IF(LENB.LE.3) LENH=2
        IF(LENB.LE.6) LENH=4
        NSYS=2
        GO TO 80
30      LENB=LENB+1
        K=K+1
        IF(BUFFER(K).NE.' '.AND.BUFFER(K).NE.'+'.AND.
       1BUFFER(K).NE.'-'.AND.BUFFER(K).NE.'*'.AND.
       1BUFFER(K).NE.'/'.AND.BUFFER(K).NE.','.AND.
       1BUFFER(K).NE.' ') GO TO 30
        IF(LENB.LE.16) GO TO 15
        NERR=NERR+1
        ERROR(NERR)=14
        LL=LENB-16
        LENB=16
```

```
      II=II+LL
15    IF(LENB.LE.8) LENH=2
      IF(LENB.GT.8) LENH=4
      NSYS=3
      GO TO 80
40    LENH=LENH+1
      K=K+1
      IF(BUFFER(K).NE.' '.AND.BUFFER(K).NE.'+'.AND.
     1BUFFER(K).NE.'-'.AND.BUFFER(K).NE.'*'.AND.
     1BUFFER(K).NE.'/'.AND.BUFFER(K).NE.','.AND.
     1BUFFER(K).NE.' ') GO TO 40
      NSYS=4
      IF(LENH.LE.2) GO TO 80
      NERR=NERR+1
      ERROR(NERR)=14
      LL=LENH-2
      LENH=2
      II=II+LL
80    RETURN
      END
```

```
      SUBROUTINE SCHSYM(IEFLG,M,SYM,HEXV)
      INTEGER ERROR(10)
      BYTE SYM(6),HEXV(4),SYMT(100,6),TEMPT(6),TEMPA(4)
      BYTE BUFFER(80),SYMA(100,4)
      COMMON /D/BUFFER,SYMA,SYMT/F/IFLAG/L/ERROR,NERR
      IF(M.LE.2) GO TO 10
      IF(IFLAG.EQ.M) GO TO 10
      LOOP=1
      DO 70 II=1,M-2
      IF(LOOP.EQ.0) GO TO 180
      LOOP=0
      DO 100 I=1,M-II-1
      IF(SYMT(I,1).EQ.SYMT(I+1,1))GO TO 20
      IF(SYMT(I,1).LT.SYMT(I+1,1)) GO TO 100
60    DO 30 JJ=1,6
      TEMPT(JJ)=SYMT(I,JJ)
      SYMT(I,JJ)=SYMT(I+1,JJ)
      SYMT(I+1,JJ)=TEMPT(JJ)
30    CONTINUE
      DO 40 JJ=1,4
      TEMPA(JJ)=SYMA(I,JJ)
      SYMA(I,JJ)=SYMA(I+1,JJ)
      SYMA(I+1,JJ)=TEMPA(JJ)
40    CONTINUE
      LOOP=1
      GO TO 100
20    DO 80 K=2,6
      IF(SYMT(I,K).EQ.SYMT(I+1,K).AND.SYMT(I,K).NE.' ')
     1 GO TO 80
      IF(SYMT(I,K).GT.SYMT(I+1,K)) GO TO 60
      GO TO 100
80    CONTINUE
100   CONTINUE
70    CONTINUE
180   IFLAG=M
      IF(IEFLG.EQ.0) RETURN
10    ZM=0
      NN=0
      DM=M-1
      MM=M-1
110   CONTINUE
```

```
      EN=ZM+(DM-ZM)/2+.5
D     WRITE(7,210) ZM,DM,EN
D210  FORMAT(1X,3F10.2)
      N=EN
D     WRITE(7,200) N,SYM(1),SYMT(N,1)
D200  FORMAT(1X,I3,2A1)
      IF(SYM(1).LT.SYMT(N,1)) DM=N
      IF(SYM(1).GT.SYMT(N,1)) ZM=N
      IF(NN.EQ.N) GO TO 65
      NN=N
      IF(SYM(1).EQ.SYMT(N,1)) GO TO 190
      GO TO 110
 190  MM=N
      NUM=MM
      I=1
 130  I=I+1
D     WRITE(7,200) I,SYM(I),SYMT(MM,I)
      IF(I.EQ.7) GO TO 160
      IF(SYM(I).EQ.' '.AND.SYMT(MM,I).EQ.' ') GO TO 160
 120  IF(SYM(I).NE.SYMT(MM,I)) MM=MM+1
      IF(MM.EQ.M) GO TO 65
      IF(SYM(I).EQ.SYMT(MM,I).AND.SYM(I-1).EQ.
     1SYMT(MM,I-1)) GO TO 130
      IF(SYM(I-1).EQ.SYMT(MM,I-1)) GO TO 120
      MM=NUM-1
      I=1
 140  I=I+1
D     WRITE(7,200) I,SYM(I),SYMT(MM,I)
      IF(I.EQ.7) GO TO 160
      IF(SYM(I).EQ.' '.AND.SYMT(MM,I).EQ.' ') GO TO 160
 150  IF(SYM(I).NE.SYMT(MM,I)) MM=MM-1
      IF(MM.EQ.0) GO TO 65
      IF(SYM(I).EQ.SYMT(MM,I).AND.SYM(I-1).EQ.
     1SYMT(MM,I-1)) GO TO 140
      IF(SYM(I-1).EQ.SYMT(MM,I-1)) GO TO 150
      GO TO 65
160   DO 170 KK=1,4
      HEXV(KK)=SYMA(MM,KK)
 170  CONTINUE
      RETURN
 65   DO 75 KK=1,4
      HEXV(KK)='0'
 75   CONTINUE
      NERR=NERR+1
      ERROR(NERR)=2
      RETURN
      END
```

```
      SUBROUTINE ERRORS
      INTEGER ERROR(10)
      COMMON /L/ERROR,NERR/M/KWRI
      WRITE(KWRI,3)
3     FORMAT(1X,'***THE ERRORS FOR THE ABOVE STATMENT
     1 ARE***')
5     GO TO (10,20,30,40,50,60,70,80,90,100,110,120,
     1130,140,150,160,170,180,190,200,210,220)
     1,ERROR(NERR)
10    WRITE(KWRI,15)
15    FORMAT(1X,'***INVALID OPCODE***')
      GO TO 2
20    WRITE(KWRI,25)
25    FORMAT(1X,'***SYMBOL NOT FOUND IN SYMBOL
     1 TABLE***')
      GO TO 2
30    WRITE(KWRI,35)
35    FORMAT(1X,'***NOT VALID SYMBOL***')
      GO TO 2
40    WRITE(KWRI,45)
45    FORMAT(1X,'***INVALID USE OF ACCUMULATOR***')
      GO TO 2
50    WRITE(KWRI,55)
55    FORMAT(1X,'***INVALID OPCODE,USE OF ACCUMULATOR
     1 OR ADDRESSING MODE***')
      GO TO 2
60    WRITE(KWRI,65)
65    FORMAT(1X,'***CANNOT USE IMMEDIATE ADDRESSING
     1 FOR THIS OPCODE***')
      GO TO 2
70    WRITE(KWRI,75)
75    FORMAT(1X,'***CANNOT USE DIRECT ADDRESSING
     1 FOR THIS OPCODE***')
      GO TO 2
80    WRITE(KWRI,85)
85    FORMAT(1X,'***CANNOT USE INDEX ADDRESSING
     1 FOR THIS OPCODE***')
      GO TO 2
90    WRITE(KWRI,95)
95    FORMAT(1X,'***CANNOT USE RELATIVE ADDRESSING
     1 FOR THIS OPCODE***')
      GO TO 2
```

```
100     WRITE(KWRI,105)
105     FORMAT(1X,'***CANNOT USE EXTENDED ADDRESSING
        1 FOR THIS OPCODE***')
        GO TO 2
110     WRITE(KWRI,115)
115     FORMAT(1X,'***CAN ONLY HAVE CHARACTER OF THE
        1 ASCII CHARACTER SET',
        1/,1X,'WITH HEXADECIMAL VALUE FROM 20 TO 5F***')
        GO TO 2
120     WRITE(KWRI,125)
125     FORMAT(1X,'***BRANCH OUT OF BOUND***')
        GO TO 2
130     WRITE(KWRI,135)
135     FORMAT(1X,'***OPERAND FIELD GREATER THAN 255
        1***')
        GO TO 2
140     WRITE(KWRI,145)
145     FORMAT(1X,'***OPERAND FIELD GREATER THAN 65535
        1***')
        GO TO 2
150     WRITE(KWRI,155)
155     FORMAT(1X,'***OPERAND NOT DECIMAL NUMBER***')
        GO TO 2
160     WRITE(KWRI,165)
165     FORMAT(1X,'***OPERAND NOT OCTAL NUMBER***')
        GO TO2
170     WRITE(KWRI,175)
175     FORMAT(1X,'***OPERAND NOT BINARY NUMBER***')
        GO TO  2
180     WRITE(KWRI,185)
185     FORMAT(1X,'***OPERAND NOT HEX NUMBER***')
        GO TO 2
190     WRITE(KWRI,195)
195     FORMAT(1X,'***NO OPERAND***')
        GO TO 2
200     WRITE(KWRI,205)
205     FORMAT(1X,'***COUNT GREATER THAN 255***')
        GO TO 2
210     WRITE(KWRI,215)
215     FORMAT(1X,'***ADDRESS GREATER THAN FFFF***')
        GO TO 2
220     WRITE(KWRI,225)
225     FORMAT(1X,'***SYMBOL ALREADY ASSIGNED VALUE
        1***')
2       ERROR(NERR)=0
        NERR=NERR-1
        IF(NERR.EQ.0) RETURN
        GO TO 5
        END
```

```
      SUBROUTINE PSYMT(M)
      BYTE SYMT(100,6),SYMA(100,4),BUFFER(80),SYMB(80)
      COMMON /D/BUFFER,SYMA,SYMT/M/KWRI
      WRITE(KWRI,10)
10    FORMAT(1X,'SYMBOL TABLE')
      M=M-1
      JJ=0
      K=1
      KK=5
50    IF(M.LT.KK) KK=M
      DO 45 I=1,80
      SYMB(I)=' '
45    CONTINUE
      DO 40 I=K,KK
      DO 20 II=1,6
      JJ=JJ+1
      SYMB(JJ)=SYMT(I,II)
20    CONTINUE
      JJ=JJ+2
      DO 30 II=1,4
      JJ=JJ+1
      SYMB(JJ)=SYMA(I,II)
30    CONTINUE
      JJ=JJ+2
40    CONTINUE
      IF(M.EQ.KK) GO TO 60
      K=KK+K
      KK=KK+5
      WRITE(KWRI,80) (SYMB(L),L=1,79)
      JJ=0
      GO TO 50
60    WRITE(KWRI,80) (SYMB(L),L=1,79)
80    FORMAT(1X,79A1)
90    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE CKLABL(SYM)
      BYTE SYM(6)
      INTEGER ERROR(10)
      COMMON /L/ERROR,NERR
      I=1
      IF(SYM(I).LT.'A'.OR.SYM(I).GT.'Z') GO TO 10
      IF(SYM(I).EQ.'X'.AND.SYM(I+1).EQ.' ') GO TO 10
      IF(SYM(I).EQ.'A'.AND.SYM(I+1).EQ.' ') GO TO 10
      IF(SYM(I).EQ.'B'.AND.SYM(I+1).EQ.' ') GO TO 10
      IF(SYM(I).EQ.'X'.AND.SYM(I+1).EQ.'      ') GO TO 10
      IF(SYM(I).EQ.'A'.AND.SYM(I+1).EQ.'      ') GO TO 10
      IF(SYM(I).EQ.'B'.AND.SYM(I+1).EQ.'      ') GO TO 10
      DO 20 I=2,6
      IF(SYM(I).GE.'0'.AND.SYM(I).LE.'9') GO TO 20
      IF(SYM(I).GE.'A'.AND.SYM(I).LE.'Z') GO TO 20
      IF(SYM(I).EQ.' ') RETURN
      GO TO 10
20    CONTINUE
      RETURN
10    NERR=NERR+1
      ERROR(NERR)=3
      RETURN
      END
```

```
      SUBROUTINE MOVI(I)
      INTEGER ERROR(10)
      BYTE BUFFER(80)
      COMMON /D/BUFFER/L/ERROR,NERR
10    I=I+1
      IF(I.EQ.80) GO TO 20
      IF(BUFFER(I).EQ.' '.OR.BUFFER(I).EQ.'   ') GO TO 10
      I=I-1
      RETURN
20    NERR=NERR+1
      ERROR(NERR)=19
      RETURN
      END
```

```
      SUBROUTINE OPSCH(OPS,HEXOP)
      INTEGER ERROR(10)
      BYTE OPTAB(197,5),OPS(5),OPHEX(197,2),HEXOP(2)
      COMMON /G/OPTAB,OPHEX/L/ERROR,NERR
      WRITE(7,200)
 200  FORMAT(1X,'OPSCH')
      WRITE(7,220) (OPS(L),L=1,5)
 220  FORMAT(1X,5A1)
      DO 20 I=1,197
      IF(OPS(1).EQ.OPTAB(I,1).AND.OPS(2).EQ.OPTAB(I,2).AND.
     1OPS(3).EQ.OPTAB(I,3).AND.OPS(4).EQ.OPTAB(I,4).AND.
     1OPS(5).EQ.OPTAB(I,5)) GO TO 30
20    CONTINUE
      NERR=NERR+1
      ERROR(NERR)=5
      HEXOP(1)='0'
      HEXOP(2)='0'
      RETURN
30    HEXOP(1)=OPHEX(I,1)
      HEXOP(2)=OPHEX(I,2)
      RETURN
      END
```

```
      SUBROUTINE NAM
      BYTE BUFFER(80)
      COMMON /D/BUFFER/M/KWRI
      WRITE(KWRI,10) (BUFFER(K),K=1,80)
 10   FORMAT(15X,80A1)
      RETURN
      END
```

```
      SUBROUTINE HEXBIN(HEXC,BIN)
      INTEGER HEXB(15,4),BIN(16)
      BYTE HEXC(4)
      COMMON /C/HEXB
      M=4
      K=1
      DO 10 I=1,M
      IF(HEXC(I).EQ.'0') GO TO 20
D     WRITE(7,100) (HEXB(J,KK),KK=1,4)
D100  FORMAT(1X,4I2)
      IF(HEXC(I).GE.'A') J=HEXC(I)-55
      IF(HEXC(I).LE.'9') J=HEXC(I)-48
      DO 50 JJ=1,4
      BIN(K)=HEXB(J,JJ)
      K=K+1
 50   CONTINUE
      GO TO 10
 20   DO 40 II=1,4
      BIN(K)=0
      K=K+1
 40   CONTINUE
 10   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE TOCOMP(BIN)
      INTEGER BIN(16)
      M=16
      DO 10 I=1,M
      IF(BIN(I).EQ.0) GO TO 40
      BIN(I)=0
      GO TO 10
 40   BIN(I)=1
 10   CONTINUE
      DO 20 J=M,1,-1
      BIN(J)=BIN(J)+1
      IF(BIN(J).EQ.1) GO TO 30
      BIN(J)=0
 20   CONTINUE
 30   RETURN
      END
```

```
      SUBROUTINE DECHEX(NUM,HEXR,M)
      INTEGER ERROR(10)
      REAL NUM,NU,KNT
      BYTE HEXR(4)
      COMMON /L/ERROR,NERR
      IF(NUM.GT.65535.) GO TO 80
      M=4
      NU=NUM
      M=4
      KNT=0
10    IF(NU.LT.16) GO TO 20
      NU=NU-16
      KNT=KNT+1
      GO TO 10
20    IF(NU.LE.9) HEXR(M)=48+NU
      IF(NU.GE.10) HEXR(M)=55+NU
      M=M-1
      IF(M.EQ.0) GO TO 70
      IF(KNT.LT.16) GO TO 50
      NU=KNT
      KNT=0
      GO TO 10
50    IF(KNT.LE.9) HEXR(M)=48+KNT
      IF(KNT.GE.10) HEXR(M)=55+KNT
60    M=M-1
      IF(M.EQ.0) GO TO 70
      HEXR(M)='0'
      GO TO 60
70    RETURN
80    DO 90 LL=1,4
      HEXR(LL)='0'
90    CONTINUE
      NERR=NERR+1
      ERROR(NERR)=21
      RETURN
      END
```

```
      SUBROUTINE HEXDEC(HEX4,NU,M)
      REAL NUM,NU,K
      INTEGER ERROR(10)
      BYTE HEX4(4)
      COMMON /L/ERROR,NERR
D     WRITE(7,220)
D 220 FORMAT(1X,'HEXDEC')
      M=4
D     WRITE(7,200) (HEX4(I),I=1,M)
D 200 FORMAT(1X,4A1)
      NU=0
      K=16
      IF(HEX4(M).GE.'0'.AND.HEX4(M).LE.'9') GO TO 20
      IF(HEX4(M).LT.'A'.OR.HEX4(M).GT.'F') GO TO 30
      IF(HEX4(M).GE.'A') NU=HEX4(M)-55
 20   IF(HEX4(M).LE.'9') NU=HEX4(M)-48
      DO 10 I=M-1,1,-1
D     WRITE(7,210) NU
D 210 FORMAT(1X,F10.2)
      IF(HEX4(I).GE.'0'.AND.HEX4(I).LE.'9') GO TO 40
      IF(HEX4(I).LT.'A'.OR.HEX4(I).GT.'F') GO TO 30
      IF(HEX4(I).GE.'A') NUM=HEX4(I)-55
 40   IF(HEX4(I).LE.'9') NUM=HEX4(I)-48
      NU=NU+NUM*K
      K=K*16
 10   CONTINUE
D     WRITE(7,210) NU
      RETURN
 30   NERR=NERR+1
      ERROR(NERR)=18
      NU=0
      RETURN
      END
```

```
      SUBROUTINE OCTDEC(OCTC,NU,M)
      REAL NU,K
      INTEGER ERROR(10)
      BYTE OCTC(6)
      COMMON /L/ERROR,NERR
      M=6
      K=8
      IF(OCTC(M).LT.'0'.OR.OCTC(M).GT.'7') GO TO 30
      NU=OCTC(M)-48
      DO 10 I=M-1,1,-1
      IF(OCTC(I).LT.'0'.OR.OCTC(I).GT.'7') GO TO 30
      X=OCTC(I)-48
      NU=NU+X*K
      K=K*8
10    CONTINUE
      RETURN
30    NERR=NERR+1
      ERROR(NERR)=16
      NU=0
      RETURN
      END
```

```
      SUBROUTINE BINDEC(BIN,M,NU)
      REAL NU,K
      INTEGER BIN(16),ERROR(10)
      COMMON/L/ERROR,NERR
D     WRITE(7,200) (BIN(L),L=1,16)
D200  FORMAT(1X,16I2)
      K=2
      IF(BIN(M).LT.0.OR.BIN(M).GT.1) GO TO 30
      NU=BIN(M)
      DO 10 I=M-1,1,-1
      IF(BIN(I).LT.0.OR.BIN(I).GT.1) GO TO 30
      X=BIN(I)
      NU=NU+X*K
      K=K*2
10    CONTINUE
      RETURN
30    NERR=NERR+1
      ERROR(NERR)=17
      NU=0
      RETURN
      END
```

```
      SUBROUTINE DECCOV(DEC,NU)
      REAL NU,K
      INTEGER ERROR(10)
      BYTE DEC(5)
      COMMON /L/ERROR,NERR
D     WRITE(7,200)
D200  FORMAT(1X,'DECCOV')
      M=5
      K=10
      IF(DEC(M).LT.'0'.OR.DEC(M).GT.'9') GO TO 30
      NU=DEC(M)-48
      DO 10 I=M-1,1,-1
      IF(DEC(I).LT.'0'.OR.DEC(I).GT.'9') GO TO 30
      X=DEC(I)-48
      NU=NU+X*K
      K=K*10
10    CONTINUE
      RETURN
30    NERR=NERR+1
      ERROR(NERR)=15
      NU=0
      RETURN
      END
```

FOOTNOTES

[1]Ron Bishop, Basic Microprocessors and the 6800 (Rochelle Park, N. J.: Hayden Book Company, Inc., 1979), p. 72.

[2]Motorola, Inc., M6800 Microprocessor Programming Manual Schaumburg, Ill.: Motorola, Inc., 1976), p. 2-8.

BIBLIOGRAPHY

Barron, David William. Assemblers and Loaders. London: MacDonald, 1969: New York: American Elsevier, 1969.

Bishop, Ron. Basic Microprocessors and the 6800. Rochelle Park, N. J.: Hayden Book Company, Inc., 1979.

Corbato, Poduska Saltzer. Advanced Computer Programming. Cambridge: MIT Press, 1966.

Day, Colin A. Fortran Techniques with Special Reference to Non-Numerical Applications. Cambridge: Cambridge University Press, 1972.

Flores, Evan. Assemblers and BAL. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1971.

Georgiou, Christos. "A Generalized Assembler." Masters Thesis, University of Illinois, Urbana, Illinois, 1969.

Jordan, Carolyn Elizabeth. "A Simulator for the Motorola 6800 Microprocessor." Florida Technological University, Orlando, Florida, 1974.

Lee, John A. N. The Anatomy of a Compiler. 2nd ed. New York: Van Nostrand Reinhold Co., 1974.

Motorola, Inc. M6800 Microprocessor Programming Manual. Schaumburg, Ill.: Motorola, Inc., 1976.